ECG ja sen artefaktien korjaus algoritmisin menetelmin

Jyrki Aho

Haaga-Helia ammattikorkeakoulu
Amk-opinnäytetyö
2023
Tradenomin tutkinto

Tiivistelmä

Tekijä(t)
Jyrki Aho

Tutkinto
Tradenomin tutkinto

Raportin/Opinnäytetyön nimi
ECG ja sen artefaktien korjaus algoritmisin menetelmin

Sivu- ja liitesivumäärä
40 + 7

Työn tarkoituksena oli tutkia, kuinka liikkuvalta kohdehenkilöltä voidaan kerätä ECG dataa usean sensorin avulla, sekä miten tämän kerätyn datan avulla voidaan kouluttaa neuroverkko korjaamaan ECG -käyriin liittyviä artefakteja. Eräs työn haasteista oli se, että sensoreita ei voitu synkronoida keskenään, minkä vuoksi oli kehitettävä uusi synkronointijärjestelmä. Ihmisen kehon koostumus ja sensorien sijoittelu vaikuttaa jonkin verran ECG -käyrien viiveisiin ja käyrän muotoihin, mikä sekin oli korjattava.

ECG- ja liikeanturidataa kerättiin kesällä 2021, jolloin suoritettiin erilaisia aktiviteetteja. Neuroverkkoa kouluttaessa käytettiin apuna myös Physionetistä saatavilla olevaa artefakti tietokantaa. Tämä ratkaisu osoittautui oikeaksi ja sen avulla neuroverkko onnistuttiin kouluttamaan tunnistamaan erilaisia ECG -käyriä.

Neuroverkko suunniteltiin mahdollisimman yksinkertaiseksi ja pieneksi, jotta sen voisi tarpeen vaatiessa siirtää esimerkiksi matkapuhelimeen. Lisäksi datan rajattu saatavuus oli myös yksi syy pitää neuroverkko rajatun kokoisena. Tämä kyseinen ratkaisu osoittautui tutkimuksen aikana oikeaksi ratkaisuksi, koska ilmeni että artefaktien korjaus kärsii, jos neuroverkon kasvattaa liian suureksi.

Työn lopussa vertaillaan valmiiden Python kirjastojen ECG -filtterien tuottamia tuloksia, sekä verrataan niitä kehitettyyn neuroverkkoon. Vertailtavat mallit perustuvat signaalinkäsittelyyn, koska yleisesti saatavilla olevat ECG -käyriin liittyvät neuroverkkomallit on suunniteltu analysoimaan ECG -käyriä. Vertailussa ilmeni, että kehitetty neuroverkko kykenee korjaamaan ECH-käyriin liittyviä artefakteja hieman paremmin, kuin löydetyt algoritmit, jotka oltiin varta vasten suunniteltu korjaamaan ECG -käyrien artefakteja.

Asiasanat
ECG, EKG, sydänkäyrä, sydänfilmi, neuroverkko, sensori, synkronointi, Python

Sisällys

1 Johdanto

Tehdessäni työharjoittelua Helsingin yliopiston kasvatustieteellisessä tiedekunnassa, olin kehittämässä uutta datan keräysjärjestelmää mobiililaitteisiin, koska käytössä ollut testiohjelmisto oli elinkaarensa ehtoopäässä. Eräs osa tätä projektia oli kehittää tähän ohjelmaan lisäosa, jonka tarkoituksena oli kerätä Movesensen sensoreista ECG -käyriä puhelimen muistiin ja lähettää nämä tiedot palvelimelle. Olin myös kiinnostunut neuroverkoista, joita voitaisiin kuvailla tietokoneella toteutetuksi älykkääksi hermoverkostoksi, jota voidaan opettaa ja joka oppimansa perusteella osaa muodostaa ennusteita syötetyn tiedon perusteella. Sain ajatuksen yhdistää nämä molemman osa -alueet opinnäytetyössäni. Helsingin yliopiston suostumuksella lainasin heiltä kolme Movesensen sensoria opinnäytetyötäni varten.

Koska useimmat käyttävät sykevöitä urheilun yhteydessä ja koska urheiluun liittyvä liikehdintä aiheuttaa häiriöitä sydämen ECG -käyrässä, minkä vuoksi päädyin tutkimaan näiden häiriöiden poistamista koulutetun neuroverkon avulla. Tämä häiriöiden poisto tuo myös hyötyä tavallisille kuntourheilijoille, koska he pääsevät näkemään parempilaatuista dataa, eivätkä he joudu tyytymään pelkkään sykevaihtelun tarkasteluun. Myös urheiluvalmentajat voisivat hyötyä tällaisesta neuroverkosta, koska parempilaatuisen datan avulla he kykenevät arvioimaan pelaajien kuntoa ja palautumista.

Opinnäytetyöni tavoitteena oli tutkia, miten usealla sensorilla voidaan kerätä ECG dataa ja miten sen avulla voidaan kouluttaa neuroverkko poistamaan ECG -käyrissä esiintyviä häiriöitä. Suoritin tämän tarkastelun Python ohjelmointikielellä ja vertailin lopputulosta olemassa oleviin ECG -käyrien puhdistus ja suodatin -algoritmeihin, jotka voidaan helposti asentaa Pythoniin.

1.1.1 Käsitteistö

Amplitudi	Aaltoliikkeen värähdysliikkeen laajuus, eli toisin sanoen suurimman aallon voimakkuuden ja nollatason erotuksena. (Story of mathematics, 2023.)
Android	Matkapuhelimissa käytettävä käyttöjärjestelmä, joka pohjautuu Linux -käyttöjärjestelmään. Windows, Linux ja MacOS ovat tyypillisimpiä tietokoneiden käytössä olevia käyttöjärjestelmiä (Helsingin yliopisto, 2023.).
Android Studio	Android Studio on ohjelmointiympäristö, jonka avulla voidaan luoda mobiilisovelluksia Android -pohjaisiin matkapuhelimiin.
Artefakti	ECG -käyrässä esiintyvä häiriö, joka voi aiheutua monista erilaisista syistä. Esimerkiksi tekokuitukangas, hengitys, liike, isku, Wi-fi, alhainen akku ja maavuoto voivat aiheuttaa häiriöitä, jotka näkyvät ECG -käyrässä perustason siirtymänä, piikkeinä tai erilaisina ECG:n voimakkaina liikahduksina. (Chanwimalueang, von Rosenberg & Mandic, 2015)
Depolarisoituminen	Depolarisoituminen on prosessi, jolloin sydämen sähköinen varaus muuttuu vähemmän negatiiviseksi, jolloin varautuneet ionit kulkeutuvat sydämen solujen sisälle (Study mind, 2023.).
e	Neperin luku tai englanninkieliseltä nimeltään Euler’s number. Se on päättymätön irrationaaliluku, jonka likiarvo on 2,71828. (Math is fun, 2023; Espoon matikkamaa, 2020)
ECG	Lyhenne sanoista electrocardiogram. ECG -käyrällä tarkoitetaan käyrää, jolla kuvataan sydämen sähköistä toimintaa. Tämä on juuri se käyrä, joka näkyy erilaisissa sairaalasarjoissa, kun potilasta hoidetaan teho-osastolla.
EKG	Lyhenne sanoista elektrokardiogrammi. Katso termin selitys kohdasta ECG -käyrä.

GATT	Lyhennys sanoista “Generic attribute profile.” Se on määritelmä, joka kuvaa matalaenergisten Bluetooth laitteiden kommunikointitapaa. Kyseessä on hyvin matalan tason kommunikointitapa, missä asiat määritellään bittien avulla (Townsend, 2014).
Impedanssimuutos	Sähkönjohtavuudessa tapahtunut muutos. Esimerkiksi kuiva iho johtaa huonommin sähköä kuin hikoilleen henkilön iho. Tämä johtunee siitä, että hiki sisältää ioneja, jotka auttavat sähkön ”siirtymisessä”. (Chanwimalueang, von Rosenberg & Mandic, 2015).

JSON	Lyhenne sanoista Javascript object notation. JSON on avoin ja standardoitu tiedostoformaatti, jota voidaan käyttää eri ohjelmointikielissä (Aalto yliopisto, 2019.). JSON -tiedosto alkaa ja päättyy aaltosulkeisiin, sekä sisältää kenttien nimiä ja arvoja. Kentät ja arvot on eroteltu toisistaan kaksoispiste -merkin avulla, sekä pilkulla erotellaan eri kenttien nimet ja arvot toisistaan. Esimerkki JSON -formaatista:
	{
	”etunimi”: ”Mikko”,
	”sukunimi”: ”Lahtinen”,
	”jasennumero”: ”002 334 56”
	}

Leaky relu	Neuroverkon aktivointifunktio, jonka avulla lasketaan neuroverkon yksittäisen neuronin arvo. Aluksi lasketaan edellisen kerroksen painotettu keskiarvo ja johon lisätään neuronin vakio. Tätä summaa voidaan merkitä s:llä. Tämän jälkeen lasketaan neuronin arvo leaky relu-aktivointifunktiolla eli yhtälöllä $g(s,h) = \begin{cases} s, & \text{kun s} \geq 0 \\ -s \cdot h, & \text{kun s} < 0, \end{cases}$ missä h on ennalta sovittu pieni positiivinen vakio. (PyTorch, 2023).
Mac-osoite	Lyhenne sanoista media access control. Se on laitekohtainen merkkikoodi, joka yksilöi verkkosovittimella varustetun laitteen (Kotimikro, 2019).

Max	Havaintosarjan $x_1, \ldots, x_n$ suurin arvo. (Wolfram Mathworld, 2023a.)
Mediaani	Keskimmäinen luku. Jos kuvitellaan meillä olevan n kappaletta havaintoja, joita voitaisiin merkitä $x_1, \ldots, x_n$. Kun nämä luvut järjestetään suuruusjärjestykseen, niin mediaani on tämän järjestetyn havaintosarjan keskimmäinen luku. Tai mikäli keskimmäisiä lukuja on kaksi kappaletta, niin silloin mediaani on keskimmäisten lukujen keskiarvo. (Wolfram Mathworld, 2023b.)
Min	Havaintosarjan $x_1, \ldots, x_n$ pienin arvo. (Wolfram Mathworld, 2023c.)
Millisekunti	Sama asia kuin tuhannesosa sekunnista.
Neuroverkko	Neuroverkko voitaisiin kuvata tietokoneella toteutetuksi älykkääksi hermoverkostoksi. Sen kouluttaminen vaatii paljon dataa ja aikaa, minkä jälkeen neuroverkko kykenee muodostamaan ennusteita uudesta datasta. Neuroverkot toisaalta säästävät ohjelmoijien aikaa, kun heidän ei esimerkiksi tarvitse kehittää äärimmäisen monimutkaisia algoritmeja tulkitsemaan valokuvien sisältöjä.
Ohjelmistokirjasto	Erillinen kokoelma erilaisia algoritmeja ja ohjelmoinnissa käytettäviä malleja, joita voidaan hyödyntää ohjelmoitaessa sovelluksia.
Orvaskesi	Ihmisen ihon päällimmäisin ja näkyvin kerros.
Painotettu keskiarvo	Lasketaan samalla tavalla kuin tavallinen keskiarvo. Ainoa ero keskiarvossa jokaisen datan painoarvo on yksi, kun taas painotetussa keskiarvossa sovitaan ennakolta mikä painoarvo asetetaan millekin datapisteelle. Tämä painoarvo voi olla positiivinen tai negatiivinen luku. Tämä ennakolta valittu painoarvo kerrotaan datapisteen arvon kanssa, minkä jälkeen näistä kerrotuista luvuista lasketaan keskiarvo. Toisin kuin Wolframin Mathwordissä (2023d) määriteltiin, niin tässä emme oleta, että painojen on oltava suurempia kuin yksi ja että niiden yhteisarvon on oltava yksi.

P -aalto	Ennen ECG:n piikkisarjaa esiintyvä pomppu ECG -käyrässä.
Python	Python on ohjelmointikieli, joka on yksinkertainen syntaksiltaan ja jota voidaan käyttää useissa erilaisissa ohjelmistoprojekteissa.
QRS-kompleksi	QRS -kompleksilla viitataan ECG -käyrässä esiintyvään voimakkaaseen piikkisarjaan, joka esiintyy sydämen ”sykähtäessä”. Q viittaa ensimmäisen-, R toisen- ja S kolmannen piikin kärkeen.
Relu	Neuroverkon aktivointifunktio, jonka avulla lasketaan neuroverkon yksittäisen neuronin arvo. Aluksi lasketaan edellisen kerroksen painotettu keskiarvo ja johon lisätään neuronin vakio. Tätä summaa voidaan merkitä s:llä, jonka jälkeen lasketaan neuronin arvo relu -aktivointifunktiolla eli yhtälöllä g(s) = max{0,s}. (Brownlee, 2019; Keras, 2023e)
Repolarisoituminen	Repolarisoituminen on prosessi, jossa sydämen napaisuus muuttuu normaaliksi ja solujen sisällä olevat ionit ohjataan ulos. (Study mind, 2023.).
REST	Lyhenne sanoista Representational state transfer. REST on ohjelmallinen rajapinta, missä tietoja ja palvelupyyntöjä lähetetään JSON -muodossa. REST ei ole kommunikaatiostandardi, vaan yleinen tapa suorittaa tietojenvaihtoa palvelun ja asiakasjärjestelmien välillä (ValueFrame, 2023; Mediasignal, 2023.).
Sensori	Mittalaite, jonka tarkoituksena on kerätä ympäristöstä tietoja. Tämän tutkimuksen yhteydessä sensorit keräsivät sydänkäyriin ja liikeantureihin liittyvää tietoa.
Sigmoid	Neuroverkon aktivointifunktio, jonka avulla lasketaan neuroverkon yksittäisen neuronin arvo. Aluksi lasketaan edellisen kerroksen painotettu keskiarvo ja johon lisätään neuronin vakio. Tätä summaa voidaan merkitä s:llä, jonka jälkeen lasketaan neuronin arvo sigmoid -aktivointifunktiolla eli yhtälöllä g(s) = 1/(1+e^(-2)). (Keras, 2023e)

SQLite	Vapaassa käytössä oleva ohjelmistokirjasto, jonka avulla ohjelmoija saa käyttöönsä SQL -tietokantakielen ominaisuudet, ilman että ohjelmoijan tarvitsee asentaa palvelimia tai säätää erilaisten asetusten kanssa (SQLite, 2023).
Sydämensyke	Kuinka monta kertaa sydän sykkii minuutin aikana.
Sydänkäyrä	Suomalainen nimitys ECG -käyrälle. Katso käsite kohdasta ECG -käyrä.
Sykevaihtelu	Sykevaihtelu tarkoittaa sitä, miten sydämensykkeen tahti muuttuu ajallisesti. Esimerkiksi kävelevän urheilijan syke on erilainen kuin urheilusuorituksen aikana.
T -aalto	ECG:n piikkisarjan jälkeinen pomppu ECG -käyrässä.
Taajuuskaista	Radiotaajuuden alue, joka on varattu tiettyyn käyttöön.
Whiteboard	Whiteboard on kommunikaatiotapa, joka toimii lähes samoin kuin REST kommunikaatio. Whiteboard kommunikaatio vaatii palvelimen ja asiakasjärjestelmän olemassaoloa (Movesense, 2021).

2 Johdatus sensoriteknologiaan

Tutkimuksen tietojenkeruussa käytettiin apuna Suunnon Movesense HR+ sensoreita, jotka kiinnitetään sykevyön avulla rintakehään. Kyseisen sensori ja siihen kytketty Android ohjelmisto perustuvat avoimeen lähdekoodiin (Suunto, 2021). Alla olevassa kuvassa on nähtävissä miltä Movesensen sensorit ja sykevyö näyttävät.

Kuva 1. Movesensen sensoreita ja sykevyö

2.1 Sensorin toiminta

Suunnon (2023a) dokumentaation perusteella, heidän sensorinsa on suunniteltu siten, että Android voi muodostaa yhteyden sensoriin Bluetoothin avulla. Kommunikoinnissa voidaan käyttää GATT -palvelin/asiakas- järjestelmää, mikä on hyvin matalan tason ohjelma, jossa lähetettyjä tietoja poimitaan bittitasolla. Movesense sensorin kanssa voidaan kommunikoida myös whiteboard -tasolla, jonka toiminta on hyvin samantapaista kuin web -ohjelmoinnissa käytetty REST -rajapinta, mikä on hyvin yleinen kommunikointitapa internetissä. Esimerkiksi jotkin internetsivustot käyttävät REST -rajapintaa noutaakseen sieltä mahdollisimman nopeasti tietoja, jotka sitten esitetään asiakkaille. Movesensen whiteboard -rajapinta tarjoaa REST:stä poiketen myös sellaisen vaihtoehdon kuin subscribe, jonka avulla ohjelmoija voi tilata sensorin lähettämään jatkuvasti dataa ilman erillisiä pyyntöjä.

2.2 Matkapuhelimen rajapinta

Matkapuhelimen sovellusrajapintana voidaan käyttää joko iOS tai Android matkapuhelimen rajapintaa. Koska käytössäni oli Android matkapuhelin ohjelmoimista varten, tarkastelen sovelluksen ohjelmointia ja rajapintaa ainoastaan Android -järjestelmän näkökulmasta. Whiteboard kommunikointia varten Android Studioon on asennettava Movesenseä varten suunniteltu Mdslib -ohjelmistokirjasto (Suunto, 2023b), joka huolehtii kommunikoinnista ja yhteyden ylläpidosta sensoriin. Kyseinen kirjasto luo automaattisesti uuden yhteyden sensoriin, mikäli yhteys katkeaa hetkellisesti. Tämä yhteys pysyy yllä maksimissaan viisi tuntia, minkä jälkeen yhteys automaattisesti katkeaa. Osa Mdslib -ohjelmistokirjaston paketeista perustuu 32- ja 64 -bitin järjestelmiin. Tämä aiheuttaa ongelmia silloin, kun kyseisen kirjaston yrittää saada toimimaan Android Studio 4.2:ssa, joka automaattisesti käyttää 86 -bitin järjestelmää. Toisin sanoen näitä asetuksia on ensin muutettava Android Studiossa, jotta kyseisen ohjelmistokirjaston saa toimimaan halutulla tavalla.

Matkapuhelimen rajapinnan ohjelmoinnissa voi käyttää apuna Movesensen omaa sovellusta, joka on vapaasti saatavilla verkossa. (Suunto, 2021.) Tämän ohjelman voi ladata Android studioon ja muokata haluamaksensa, kun ryhtyy testaamaan sovelluksen ja Movesense sensorin toimintaa. Käyttäjä voi luoda myös oman ohjelman, joka voi käyttää apuna vapaasti saatavilla olevaa MDSlib -kirjastoa.

Jotta haluttu sensori löydetään, matkapuhelimen skannaa lähistöltä löytyvät Bluetooth laitteet. Sen jälkeen matkapuhelin kerää lähellä olevien laitteiden mac -osoitteet ja laitteisiin liittyvät nimet. Kaikki Movesensen sensorit on nimetty ”Movesense XXXXXXX” tyylisesti, missä X:t vastaavat numeroita. Ohjelma muodostaa Movesense laitteen kanssa yhteyden, jonka aikana sovitaan sarjanumero, minkä avulla ohjelmisto pystyy kommunikoimaan juuri tämän kyseisen sensorin kanssa. Sensorille voidaan lähettää pyyntöjä, sekä tilauksia, jolloin sensori lähettää automaattisesti keräämiään tietoja. Tällöin ohjelmiston on kuunneltava tiettyä URL -osoitetta ja poimittava sieltä tilaamiensa sensorien tiedot. Movesense sisältää myös turvaominaisuuden, jonka avulla sovellus ja sensori voidaan parittaa keskenään ennalta sovitun PIN -numeron avulla. Koska lopputyössäni tarkastellaan häiriöiden vaikutusta käyrään, en nähnyt syytä sille, miksi olisin ottanut kyseisen turvaominaisuuden käyttöön.

3 ECG

ECG eli electrocardiogram tarkoittaa sydämen sähköisen toiminnan tallentamista, joka on tuttu useimmille ihmisille erilaisista lääkärisarjoista. Sille voitaisiin käyttää myös suomalaista lyhennettä EKG eli elektrokardiogrammi tai termiä sydänkäyrä, mutta päädyin työssäni käyttämään lyhennettä ECG, koska se on lyhenteenä kansainvälisesti tunnettu. ECG:stä käytetään myös nimeä sydänfilmi, vaikka sitä ei ole koskaan tallennettukaan filmille. ECG -käyrän tarkoituksena on mitata sydämen sähköistä toimintaa ja piirtää sen toiminnasta ajallinen kuvaus. ECG:n historia lähtee liikkeelle jo 1600 -luvulta, mutta ensimmäisen modernin ECG -käyrää mittaavan laitteen rakensi Willem Eithoven, joka esitteli sen toimintaa ensimmäisessä kansainvälisessä fysiologien konferenssissa vuonna 1889. Tämä ensimmäinen mittalaite painoi noin 270 kiloa ja sensoreina toimivat suolaliuosta sisältäneet sangot, joihin upotettiin raajat. Eithoven julkaisi vuonna 1903 ECG:tä koskevan artikkelin, missä hän käytti graafisessa esityksessään kirjaimia P, Q, R, S ja T kuvaamaan käyrään liittyviä muutoksia. Nämä merkinnät ovat nykyäänkin käytössä samassa tarkoituksessa ja Eithoven sai työstään lääketieteen Nobelin palkinnon vuonna 1924. (Jenkins, 2009; NobelPrize.org, 2021; Salam, 2019; Wong, 2019; Syvänne & Hekkala, 2019.)

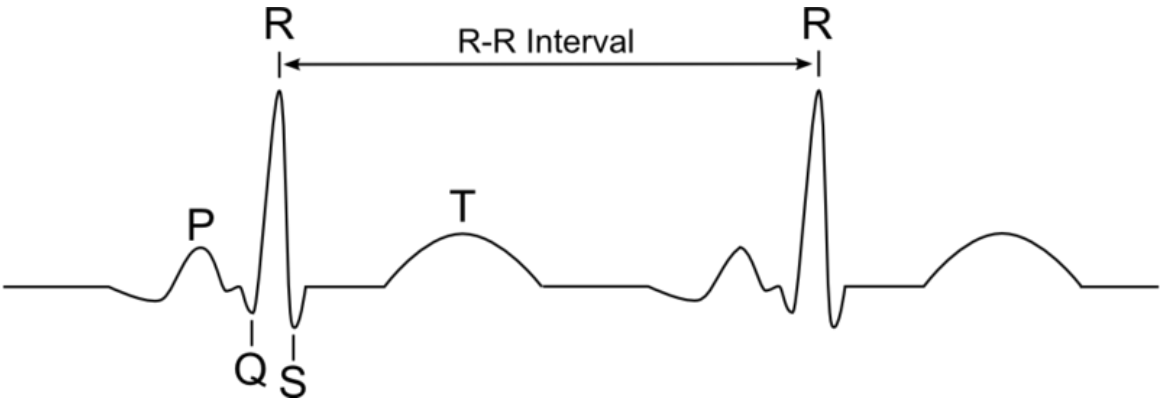
ECG:tä mittaavat laitteet ovat pienentyneet huomattavasti ja niiden mittaustarkkuudessakin on tapahtunut huomattavaa kehitystä. Nykyisin sydämen toimintaa mittaavia sensoreita löytyy jo erilaisista urheilukelloista ja jopa Oura -sormuksesta. Sydämen toiminnan mittaus on yleensä liitetty lääketieteeseen, mutta sitä voidaan käyttää apuna myös urheilussa ja nykyään jopa henkilötutkimuksissa. Urheilussa sydämen toimintaa voidaan käyttää apuna seurattaessa urheilijan rasittumista, sekä sitä millainen palautumiskyky kyseisellä urheilijalla on. Tämän tiedon perusteella analyysitiimi voi luoda uniikin treeniohjelman jokaiselle urheilijalle. Henkilötutkimuksissa puolestaan voidaan tarkkailla tutkimuskohteen stressitasoja, koska tutkimukseen liittyvät stressiin liittyvät kyselyt eivät voi antaa niin tarkkoja tuloksia kuin tutkittavan henkilön sydän. Nykyisin tiedetään esimerkiksi, että tutkittavan henkilön ikä ja sukupuoli, sekä ruumin rakenne vaikuttavat ECG -käyrään ja QRS -kompleksin volttеihin. QRS -kompleksilla tarkoitetaan ECG -käyrässä esiintyvää voimakasta piikkisarjaa, mistä kerrotaan lisää seuraavassa luvussa. Lisäksi ECG -käyrään vaikuttaa myös sensorin sijoittaminen, minkä lisäksi eri eläinlajeilla saattaa olla erilainen ECG -käyrä kuin ihmisillä. (Abdulla, Bonney, Khalid & Awad, 2016, sivu 2; Poutanen & Hiippala, 2016; Tiusanen & Pastell, 2016)

3.1 ECG -käyrän muodostuminen

Sydämen sähköiseen toimintaan liittyy lihassolujen sähköistä toimintaa, mihin liittyy sellaisia käsitteitä kuin sydänlihaksen repolarisoituminen/ depolarisoituminen, sekä Kalium ja

Natrium ionien toiminta. Lisäksi sydämen sähköinen toiminta luo sen ympärillä sähkökentän, jolla on olemassa kolmeakselinen suunta, joka on otettava huomioon, kun sensoreita sijoitetaan tutkittavaan henkilöön ja ryhdytään keräämään tietoja sydämen toiminnasta.

Sydämen ECG -käyrän muodostumista voidaan ymmärtää helpommin, kun ryhdytään tarkastelemaan ihmisen verenkiertoa ja sydämen toimintaa. Sydämen vasemmanpuoleinen eteinen pumppaa vähähappista verta keuhkoihin meneviin verisuoniin, josta uudelleen hapettunut veri palaa sydämen oikeaan eteiseen. Sydämen oikeasta eteisestä veri siirtyy oikeaan kammioon, josta hapettunut veri ohjautuu takaisin henkilön verenkiertoon. Vähähappinen veri palautuu lopulta takaisin sydämen vasempaan eteiseen, mistä se ohjautuu vasempaan kammioon, mistä se lopulta ohjautuu henkilön keuhkoihin keräämään happea. Keuhkoista happirikas veri ohjautuu henkilön oikeaan eteiseen ja kierto lähtee taas alusta liikkeelle. (NCBI, 2019; Matthews 2021)



Kuva 2. ECG -käyrän rakenne (Wikimedia, 2021).

ECG -käyrä muodostuu, kun sydämen oikea eteinen alkaa depolarisoitumaan, eli varautumaan, jolloin ECG -käyrään alkaa muodostua P -aalto. P -aalto saavuttaa huippunsa, kun sydämen väliseinä varautuu ja päättyy vasemman eteisen varautumiseen. Tämä koko P -aalto kestää yleensä alle 100 millisekuntia (ms). Tämän P -aallon aikana sydämen eteiset imevät itseensä verta. (Abdulla, Bonneym Khalid ja Awad, 2016, kappaleet 1–2; Atlas of Human Cardiac Anatomy, 2021; Goy, Staufer & Schlaepfer, 2013, kappale 1; NCBI, 2019.)

P -aallon jälkeen seuraa yleensä lyhyt, noin 20–100 millisekunnin tauko, jonka jälkeen ECG -käyrässä esiintyy noin 60–110 millisekuntia kestävä QRS -kompleksi. QRS -kompleksin aikana sydämen kammiot aktivoituvat peräkkäin. Aktivoituminen alkaa sydämen väliseinän vasemmalta puolelta, mistä aktivoituminen etenee oikealle ja eteenpäin. Q -piikki näkyy ECG -käyrässä, kun sydämen väliseinä aktivoituu. R -piikki vastaa vasemman vapaan seinän aktivoitumista ja S -piikki vastaa vasemman tyvisegmentin depolarisoitumista. QRS -kompleksin aikana sydämen läpät aukeavat R -piikin aikana, jolloin veri siirtyy sydänkammioista elimistön verenkiertoon. Oikeassa kammiossa oleva veri ohjautuu

henkilön keuhkoihin hapettumaan, kun taas vasemmassa kammiossa oleva hapekas veri siirtyy verenkiertoon. QRS -kompleksin jälkeen seuraa pieni tauko, jonka jälkeen ECG -käyrään piirtyy T -aalto. Se syntyy, kun sydämen kammiot repolarisoituvat ja kammiot rentoutuvat, jolloin sydämen eteisessä oleva veri siirtyy sydänkammioihin. (Abdulla et al., 2016, kappaleet 1–2; Atlas of Human Cardiac Anatomy, 2021; Goy, Staufer & Schlaepfer, 2013, kappale 1; NCBI, 2019.)

ECG -käyrän perusteella voidaan myös laskea henkilön syke. Sydämensyke voidaan määritellä siten, että se on 60 sekunnin aikana ilmenevien R -piikkien lukumäärä. Eli periaatteessa sydämen sykkeen voi laskea yhtälöllä

$$\text{Sydämensyke (pbm)} = \frac{60\ s}{R_i \text{ ja } R_{i+1} \text{ piikken välinen etäisyys (sekunteina)}},$$

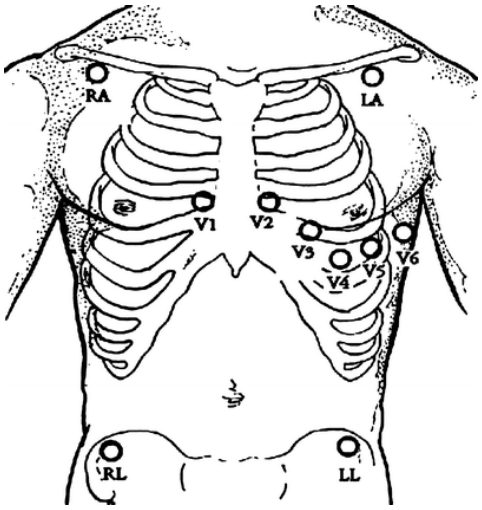
missä esiintyy kaksi peräkkäistä R -piikkiä. (Abdulla et al., 2016, sivu 45–46.) Käytännön algoritmeissa sydämensyke lasketaan siten, että sekunnit on muutettu millisekunneiksi, ja kyseinen algoritmi kerää N -kappaletta peräkkäisiä R -piikkejä, yleensä 5–8 kappaletta. Tällöin sydämensyke selvitetään laskemalla ensimmäinen ja viimeisen R -piikin ajallinen etäisyys, joka jaetaan luvulla N. Tämä laskettu näennäiskeskiarvo jaetaan 60 000 millisekunnilla (=60 sekuntia), jolloin lopputulokseksi muodostuu hieman vakaampi luku sydämensykkeeksi, koska kyseinen laskentatapa vähentää pyöristys- ja mittausvirheitä.

3.2 ECG:n mittaaminen

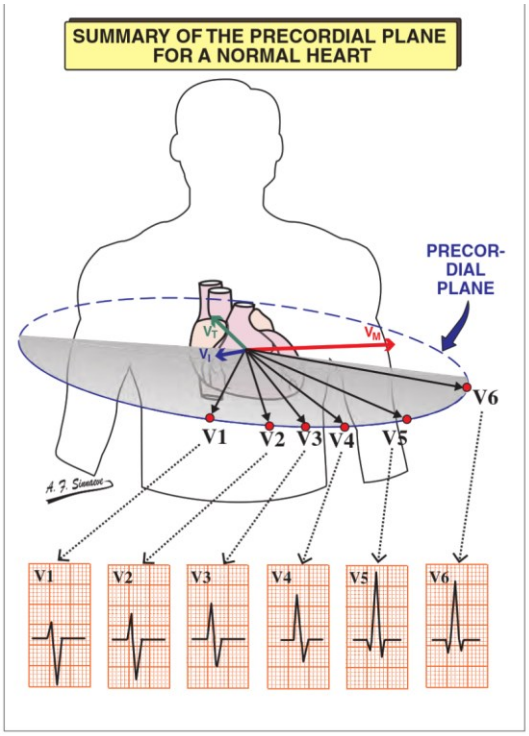
ECG tietoja kerätään yleensä sairaaloissa, jolloin laitteissa on käytössä useampi sensori ja tutkittavan potilaan on mittauksen aikana maattava aloillaan. Potilaasta pyritään tällöin keräämän noin 5–10 minuuttia laadukasta mittaustietoa potilaan sydämen toiminnasta. Tämän tutkimusasetelman ongelmana on, että vaikka tällä tavalla voidaan mitata sydämen lyönnin vaihteluita, niin kyseinen menetelmä ei kykene havaitsemaan kaikkia sydänongelmia. Tämän lisäksi diagnoosien teko on näissä tapauksissa hidasta ja virhealtista. ECG -käyriä on myös kyettävä mittamaan pitkäaikaisilta potilailta tai koehenkilöiltä lähes reaaliaikaisesti, eikä heitä voida pyytää makaamaan paikoillaan. Pitkäaikaisen potilaiden tarkkailun tarkoituksena on havainnoida sydämen toimintaa erilaisten aktiviteettien aikana ja löytää mahdollisia poikkeuksia. Käyttäjän mukavuuden kannalta laitteen keveys ja tarkkuus ovat tärkeitä tekijöitä. (Ahmed, Hilal-Alnaqbi, Hemairy & Ahmad, 2018.; Chanwimalueang, von Rosenberg & Mandic, 2015.; Ghaleb, Kamat, Salleh, Rohani & Razak, 2018.)

ECG:tä mittaavat sensorit sijoitetaan yleensä rintakehän päälle, mistä on löydettävissä kuusi hyvää mittauspaikkaa. Lisäksi ECG voidaan mitata samanaikaisesti myös henkilön

ranteista ja nilkoista. Sensorien potentiaaliset sijoituspaikat ovat nähtävissä kuvassa 3. (He, Clifford & Tarassenko, 2006; Syvänne & Hekkala, 2019.) Mitattaessa henkilön ECG:tä on otettava huomioon sensorien sijoituspaikat, koska eri paikkoihin sijoitetut ECG:tä antavat hieman erilaisen ECG -käyrän sydämen toiminnasta, riippuen siitä missä kulmassa kyseinen sensori on sydämeen nähden. Tämä ilmiö on selkeästi havaittavissa kuvassa 4, jonka Strootband, Barold ja Sinnaeve (2016) ovat luoneet. Kuvasta on nähtävillä, että R -piikki saattaa siirtyä muutaman millisekunnin myöhemmäksi ja ECG -käyrän rakenteessa on nähtävillä muutos, kun sensori sijoitetaan eri paikkoihin rintakehällä.



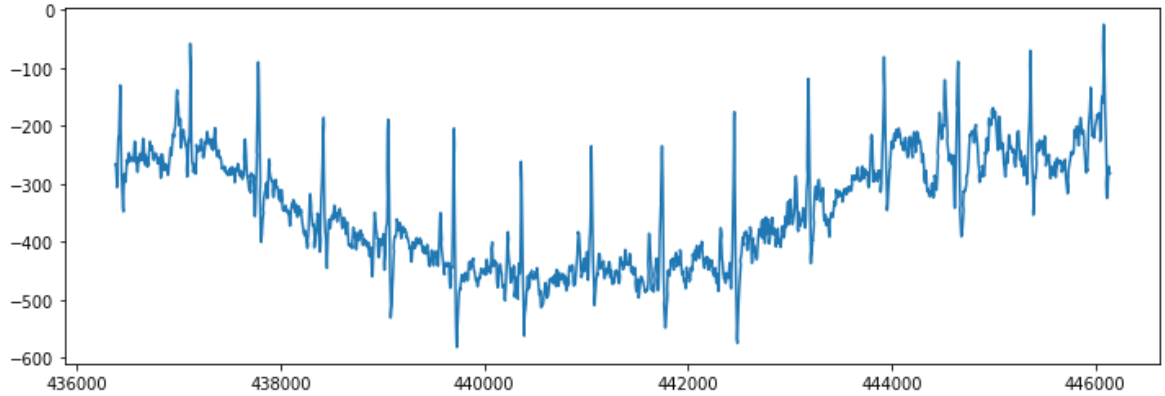
Kuva 3. Sensorien sijoittamispaikat sairaalaympäristössä. (He, Clifford & Tarassenko, 2006, sivu 107.)



Kuva 4. Sensorin sijoituspaikat vaikuttavat tallennettavan ECG:n rakenteeseen (Strootbandt, Barold & Sinnaeve, 2016, sivu 62.)

3.3 ECG:ssä esiintyvät häiriöt eli artefaktit

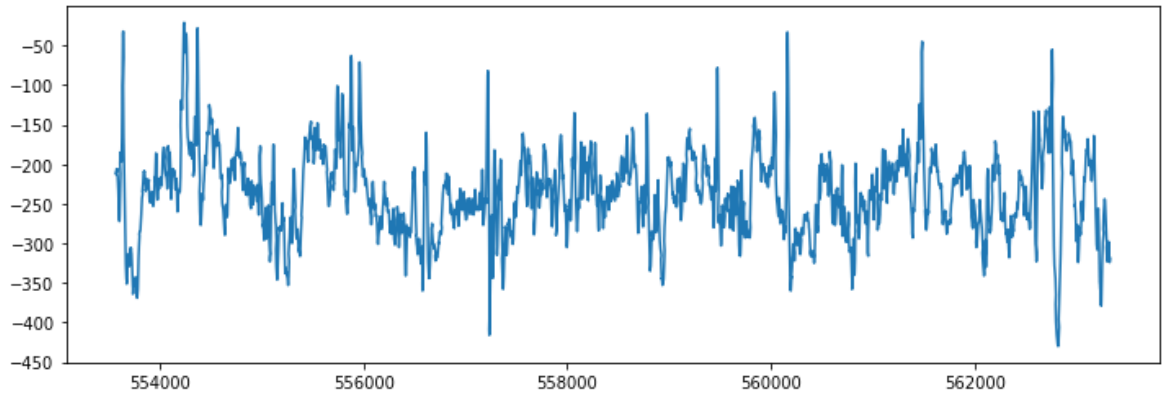
ECG -käyrien mittaamisissa esiintyy häiriöitä, joista käytetään nimitystä artefakti. Artefakteja esiintyy erilaisista syistä, kuten kehon liikkeestä, ihon impedanssimuutoksista ja muista seikoista johtuen. Erityisesti kannettavat sensorit ovat alttiita artefakteille, mikä johtuu niiden alhaisesta mittaustiheydestä ja upottamisista (Chanwimalueang, von Rosenberg & Mandic, 2015.). ECG -käyrä saattaa lähteä vaeltamaan perustasosta, mikä yleisesti aiheutuu hengityksestä tai ruumiin liikkeistä.



Kuva 5. Perustason vaeltaminen

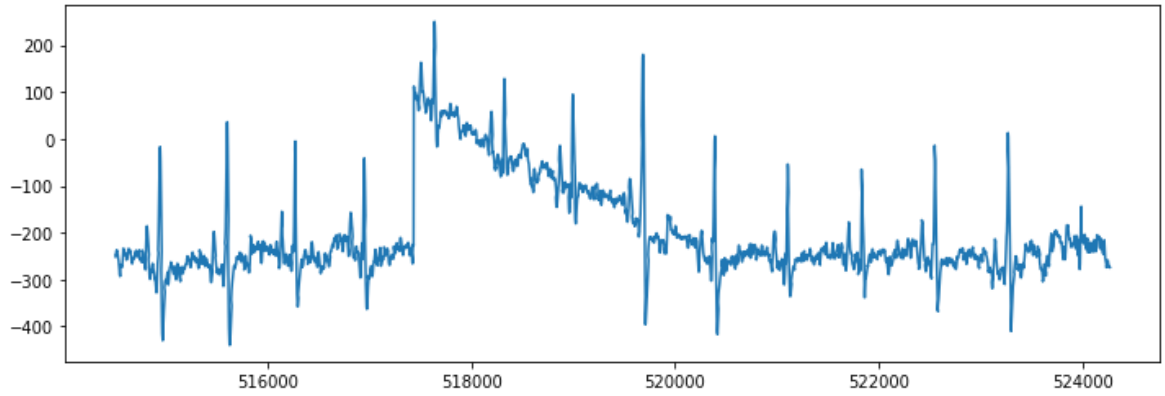
Erisuuruiset ECG -piikit voivat johtua laitteen elektronivuodosta, maadoitusongelmasta tai toisen elektronisen laitteen läheisyydestä. Koska tutkimuksessa käytetään Movesensen sensoreita, joita voi käyttää jopa uidessa, ei näiden sensoreiden pitäisi aiheuttaa tämänkaltaisia artefakteja. Tosin meidän on huomioitava elektronisten laitteiden läheisyys, minkä vuoksi matkapuhelinta on pidettävä mahdollisimman kaukana näistä sensoreista tietojen keruun aikana.

ECG -käyrässä esiintyvät lyhytkestoiset nopeat piikit johtuvat lihasten liikkeistä tai vilunväristyksistä. Liikeartefaktien poistaminen ECG -käyrästä on erittäin vaikeaa ja joskus liikeartefakti peittää täysin sydämen ECG -signaalin. Liikeartefaktien poistaminen vaikuttaa myös suoraan ECG -käyrään, mikä sekin aiheuttaa ongelmia sen tulkitsemisessa. Tämän takia, kannettavien sensorien sijoittaminen on tärkeää ja isoja lihaksia on syytä välttää. Myös jotkin taudit, kuten esimerkiksi Parkinson, saattavat aiheuttaa samankaltaisia häiriöitä ECG -käyriin. Esimerkiksi kuvassa 6 on hyvä havaintokuva, minkälaisia häiriöitä nämä sairaudet voivat aiheuttaa ECG -käyrään. (Ghaleb, Kamat, Salleh, Rohani & Razak, 2018.)



Kuva 6. Sarja lyhytkestoisia piikkejä.

ECG -käyrässä esiintyvä suuri perustason hetkellinen siirtymä aiheutuu esimerkiksi yskästä. Orvaskedessä eli ihossa tapahtuva muutos tai venytteleminen aiheuttaa myös tämänkaltaisia artefakteja. Venyttely voi aiheuttaa ECG -käyrässä hetkellisen siirtymän, joka on havainnollistettu kuvassa 7.



Kuva 7. Perustason hetkellinen siirtymä

Muita mahdollisia artefaktin lähteitä on paristojen virran vähyys tai mahdollinen katkos tietoyhteydessä. Koska Bluetooth siirtää tietoja 2,4 GHz taajuuskaistalla, on se altis esimerkiksi Wi-Fi verkkojen aiheuttamalle häirinnälle. Urheiluvaatteet on yleisesti tehty tekokuidusta, jonka liike saattaa myös häiritä joidenkin sensorien toimintaa. Lisäksi sydämen sähköakselin suunta voi muuttua erilaisissa sydänolosuhteissa, kuten esimerkiksi kammion suurentumassa. (Abdulla, Bonney, Khalid & Awad, 2016; Sivaraks & Ratanamahatana, 2015.; Debbabi, Asmi & Arfa, 2010.)

3.4 ECG -algoritmit ja rakenne

Kun ECG:n artefakteja ryhdyttiin tutkimaan ja eliminoimaan 40 -vuotta sitten, tarkasteltiin aluksi ajallista tai avaruudellista keskiarvotekniikkaa. Ajallisella keskiarvoistamisella tarkoitetaan, että laskemme ECG havainnoista keskiarvoja ennalta sovittujen aikavälien puitteissa. Ajallinen keskiarvoistaminen vaatii paljon kerättyä tietoja, jotta sen avulla voitaisiin

tehokkaasti poistaa artefakteja tiedoista. Avaruudellisella keskiarvotekniikalla tarkoitetaan puolestaan sitä, että hyvin pienelle alalle asetetaan useita sensoreita keräämään ECG dataa ja näistä havainnoista lasketaan keskiarvo, jonka oletetaan siten kuvaavan hyvin ECG -käyrän arvoja. Avaruudellisessa keskiarvotekniikassa ongelman aiheuttavat fysikaaliset rajoitteet, koska samaan kohtaan ei voida asettaa kovinkaan montaa elektrodia mittaamaan sydämen sykettä. Näitten rajoitteitten takia tutkijat ryhtyivät tutkimaan erilaisia filttereitä ja algoritmisia menetelmiä, joiden avulla pyrittiin vähentämään artefaktien vaikutusta, mutta jokaisessa menetelmässä on kuitenkin omat puutteensa. Tietokoneiden kehittymisen vuoksi laskentatehot ja muisti eivät tuota nykyisin ongelmia, kun prosessoidaan ECG -käyriä. On kuitenkin huomattava, että läheskään kaikkia käytössä olevia algoritmeja ei voida käyttää pattereilla toimivissa laitteissa, koska kyseisten laitteiden laskentateho ja muisti ovat erittäin rajoitettuja. Edistyneitä algoritmeja on testattu esimerkiksi Shimmerin sulautetuissa sensoreissa, jolloin todettiin laitteen käyttöajan lyhentyvän. Tämän takia ECG:tä mittaavat sensorit yhdistetään matkapuhelimen sovellukseen tai johonkin muuhun ohjelmistoon, jotka analysoivat langattomasti kerätyn ECG -käyrän tiedot. (He, Clifford & Tarassenko, 2006.; Köhler, Henning & Orglmeister, 2002.; Mazomenos, Biswas, Acharyya, Chen, Maharatna, Rosengarten, Morgan & Curzen, 2013.)

ECG -käyrissä esiintyvä piikki eli QRS -kompleksi, on sen kaikista tunnistettavin muoto. Tämän vuoksi sitä käytetään yleisesti erilaisten automaattisten ratkaisujen pohjana sydämen sykkeen tarkastelussa, sydämen kierron tunnistamisessa ja jopa ECG tietojen pakkaamisessa. Nykyisin QRS -kompleksin löytämisessä käytetään apuna erilaisia geneettisiä algoritmeja, aallon muuttajia, heuristisia menetelmiä, suodatinpankkeja ja neuroverkkoja. Näiden algoritmien kehitys on johtanut myös joidenkin uusien termien, kuten esimerkiksi false R -detection, käyttöönottoon, millä viitataan niihin ECG -käyrissä esiintyviin piikkeihin, jotka algoritmi tunnistaa R -piikeiksi, mutta jotka eivät todellisuudessa ole niitä. (Köhler, Henning & Orglmeister, 2002.)

ECG:tä tutkiva algoritmi voidaan jakaa karkeasti kahteen vaiheeseen. Näistä ensimmäistä sanotaan esikäsittelyvaiheeksi, missä ECG -sensoridataa pyritään keräämään 256 Hz – 512 Hz vauhdilla, koska se takaa hyvälaatuisen ECG -käyrän keruun. Tämän jälkeen ECG suodattaa erilaisten lineaaristen ja epälineaaristen algoritmien avulla, jotta signaalin seasta saataisiin paremmin poimittua QRS -kompleksi, joka on kooltaan ainoastaan 10 Hz – 25 Hz. Lisäksi nopea datankeruu auttaa poistamaan perustason vaelluksen signaalista. ECG -käyriin sovelletaan usein high -pass ja low -pass suodatusta, joka tuo paremmin esiin QRS -kompleksin, sekä samalla poistaa häiriöitä käyrältä. Debbabin, Asmin ja Arfan (2010) mukaan QRS -kompleksi löydetään parhaiten, kun ECG -käyrä saadaan suoristettua x -akselin suuntaiseksi. Näiden suodatinten huonona puolena on se, että se ei pysty

poistamaan saman taajuisia artefakteja datasta. Tämän takia esikäsittelyvaiheessa sovelletaan erilaisia strategioita poistamaan artefakteja käyrästä. Eräitä menetelmiä on käyttää keskiarvomenetelmää tai mediaanimenetelmää, joiden tarkoituksena on jakaa data liikkuviin aikaikkunoihin ja oikaista sen muoto siten, että datasta saadaan poistettua sellaiset artefaktit kuin perustason vaeltaminen ja hetkelliset siirtymän aiheuttamat vaikutukset. (Köhler, Henning & Orglmeister, 2002.; He, Clifford & Tarassenko, 2006.; Tobón-Cardona, Kenttä, Porthan, Tikkanen, Oikarinen, Viitasalo, Salomaa, Huikuri, Junttila & Seppänen, 2018.)

Toista vaihetta sanotaan päätösvaiheeksi. Siihen on ohjelmoitu logiikka tunnistamaan ECG -käyrän piikit ja suorittamaan päätöksen piikin luonteesta, kuten onko kyseessä oikea piikki vai virheellinen piikki. Toinen vaihe on hyvin riippuvainen esikäsittelyvaiheen tuottamasta käyrästä. Jotkin päätösvaiheen algoritmit pyrkivät myös löytämään ne R -piikit, joita käytetty algoritmi ei ole tunnistanut artefaktien vuoksi ja jotkin algoritmit pyrkivät poistamaan virheelliset R -piikit. (Köhler, Henning & Orglmeister, 2002.)

On olemassa myös algoritmeja, joita voidaan käyttää apuna niin esikäsittelyvaiheessa kuin päätösvaiheessakin. Esimerkiksi ensimmäisen ja toiseen asteen derivaattoja käytetään apuna niin esivaiheen prosessoinnissa, kuin päätösvaiheessakin. Toinen hyvin tunnettu menetelmä on neuroverkkojen käyttö, jota voidaan käyttää jo esivaiheen prosessoinnissa ja suorittamaan päätösvaiheen prosesseja. Yleensä ECG -käyriä halutaan tulkita reaaliaikaisesti, jolloin tarvitaan hyviä algoritmeja tunnistamaan ECG -käyrän rakenne, muodot ja arvot. (Köhler, Henning & Orglmeister, 2002.; Mazomenos & all, 2013.)

3.5 Pan-Tompkins -algoritmi

Ensimmäisiä ja edelleen yleisesti käytetyin ECG -algoritmi on Pan-Tompkins -algoritmi, joka esiteltiin vuonna 1985 ja joka kykenee löytämään ECG -käyrien QRS -komplekseja reaaliaikaisesti. Kyseinen menetelmä perustuu ECG -käyrien kulmakertoimen, amplitudin ja leveyden analysointimenetelmiin. Algoritmin käyttämä bandpass -suodatin auttaa tehokkaasti vähentämään artefaktien vaikutusta dataan.

Bandpass -suodatin auttaa vähentämään liikkeen, perustason vaelluksen ja T -aallon aiheuttamien artefaktien vaikutusta ECG -käyrään. Low-pass- ja High-pass suodattimet auttavat korostamaan QRS -kompleksin arvoja käyrässä. ECG -käyrästä otetun derivaatan avulla tarkastellaan kulmakertoimien vaikutusta dataan. QRS -kompleksin aikana derivaatan arvot ovat korkeimmillaan, verrattuna normaaliin ECG -käyrän arvoihin. ECG -käyrän neliöinnin (eli korottamalla toiseen potenssiin) tarkoituksena on muuttaa datan arvot posi-

tiivisiksi, minkä lisäksi neliöinti auttaa korostamaan derivaatan korkeampia arvoja. Liikkuvan aikavälin integroinnin tarkoituksena on saada muodostettua informaatiota ECG -aallon rakenteesta. Käytetyn aikavälin koko ei saa olla liian leveä, koska muussa tapauksessa aaltomuoto yhdistää QRS -kompleksit ja T -aallon arvot yhteen. Useat algoritmit esiprosessoivat samaan tapaan ECG -käyriä, mutta päätösvaiheen algoritmit poikkeavat Panin ja Tompkinsin (1985) kehittämästä algoritmista. (Pan & Tompkins, 1985.)

Pan ja Tompkins (1985) käyttivät signaalin tarkastelussa reaaliaikaista adaptiivista algoritmia, joka tarkasteli esiprosessin tuottaman datan kynnysarvoja ja laskivat sen perusteella R -piikkien sijainnit ECG -käyrällä. Debbabi ja kumppanit (2010) puolestaan käyttivät fiksattua algoritmia kynnysarvojen tarkasteluun, koska se vähensi datan prosessointiaikaa. He eivät kuitenkaan artikkelissa esittäneet tätä päätösvaiheen algoritmin rakennetta. Nair (2010) puolestaan yhdisti Pan-Tompkins- ja aallonhajottamis algoritmit, jotta hän saisi selvitettyä tehokkaammin ECG -käyrien rakenteen. Hänen algoritminsa pohjautui Pan-Tompiksin -algoritmin tehokkuuteen löytää R -piikit datasta, minkä jälkeen aallonhajottamis -algoritmia käytettiin apuna, jotta löydettäisiin myös ECG -käyrässä esiintyvät P- ja T -aallot.

4 Neuroverkot

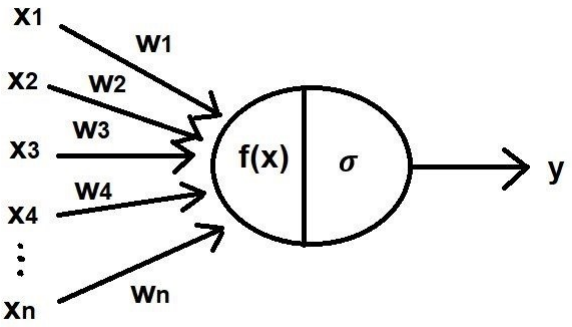
McCulloch ja Pits esittelivät vuonna 1943 ensimmäistä kertaa neuronin käsitteen, missä neuroni vastaanotti lineaarista dataa useasta pisteestä ja laski sen perusteella lähettääkö neuroni signaalin vai ei. Hebb kehitti tätä ajatusta hieman pidemmälle kirjassaan, missä hän esitti ajatuksen tällaisen neuroverkon kouluttamisesta. Näiden mallien pohjalta Frank Rosenblatt julkaisi vuonna 1957 ensimmäisen neuroverkkoja koskevan julkaisun, jossa hän käsitteli neuroverkkoja ja niiden automatisointeja. Näiden alkupäivien jälkeen neuroverkot ovat kehittyneet huimasti. (Kelleher, 2020, s. 94-104.)

Neuroverkkojen suurin ongelma on ollut, että niiden kouluttaminen vaatii paljon ennalta merkittyä dataa. 2000 -luvulla tapahtunut tietokoneiden nopeutuminen, Internetin kehittyminen ja tietoaineistojen helppo saatavuus ovat kuitenkin luoneet olosuhteet, missä neuroverkoille on saatavilla massiivinen määrä dataa ja neuroverkkoja on mahdollista rakentaa tavallisilla tietokoneilla. Koska neuroverkkoja ovat joustavia ja niitä on helppo kouluttaa ilman asiantuntemusta, ne ovat herättäneet suurta suosiota ja ovat nykyisin käytössä lähes kaikilla elämänalueilla. Niitä käytetään apuna signaalikorjauksessa, kuva ja tekstianalyyseissa tai kun datan perusteella pyritään luomaan ennusteita. (Kelleher, 2020, s. 24-29. Marini, Bucci, Magri & Magri, 2008.; Sugomori, Kaluža, Soares, Souza, Kaluza & Soares, 2017.)

4.1 Neuroverkkojen rakenne

Neuroverkko on joukko toisiinsa kytkeytyneitä funktioita, joka syötetyn datan avulla pyrkii luomaan ennusteita. Yksinkertaistetussa mallissa neuroverkon voidaan kuvitella koostuvan useista neuroneista, jotka ovat kytkeytyneitä kaikkiin seuraavan kerroksen neuroneihin. Tämä yksittäinen neuroni voi vastaanottaa tietoa yksittäisestä syötettävästä datapisteestä tai vastaanottaa tietoja useilta aikaisemman kerroksen neuroneilta, joiden perusteella neuroni kykenee laskemaan oman arvonsa aktivointifunktion avulla, joka tässä tapauksessa on sigmoid -funktio, joka tunnetaan myös logistinen funktio nimellä. Kuvassa 8 on esitetty yksittäisen neuronin rakenne. Tässä kuvassa neuroni vastaanottaa tietoja syötetystä datasta tai usealta aikaisemman kerroksen neuronilta, joiden sisältämää tietoa kuvataan arvoilla x_1, x_2, …, x_n. Neuroni suosii enemmän tietyistä pisteistä tulevia arvoja, kun taas toisista pisteistä tuleviin arvoihin neuroni suhtautuu neutraalisti tai negatiivisesti. Tätä suhtautumista aikaisemman kerroksen dataan kuvataan numeeristen painojen avulla, joita kuvassa on merkitty arvoilla w_1, w_2, …, w_n, sekä solmun omaa painoa merkitään b:n avulla. Kun summalausekkeesta käytetään merkintää s, niin logistisen sigmoid -aktivaatiofunktion σ avulla lasketaan neuronin arvo y. Sigmoid -funktio huolehtii siitä, että jokainen neuroni saa jonkin arvon nollan ja ykkösen väliltä. Yleensä neuroverkon neuronit ovat

rakenteeltaan tämänkaltaisia, mutta joissain neuroverkoissa saattaa olla käytössä muitakin aktivointifunktioita, aktivointifunktiot voivat vaihdella kerroksittain ja lisäksi saman kerroksen neuronit saattavat olla kytkeytyneitä muihin saman kerroksen neuroneihin. (Kelleher, 2020, s. 68-74; Keras, 2023e)

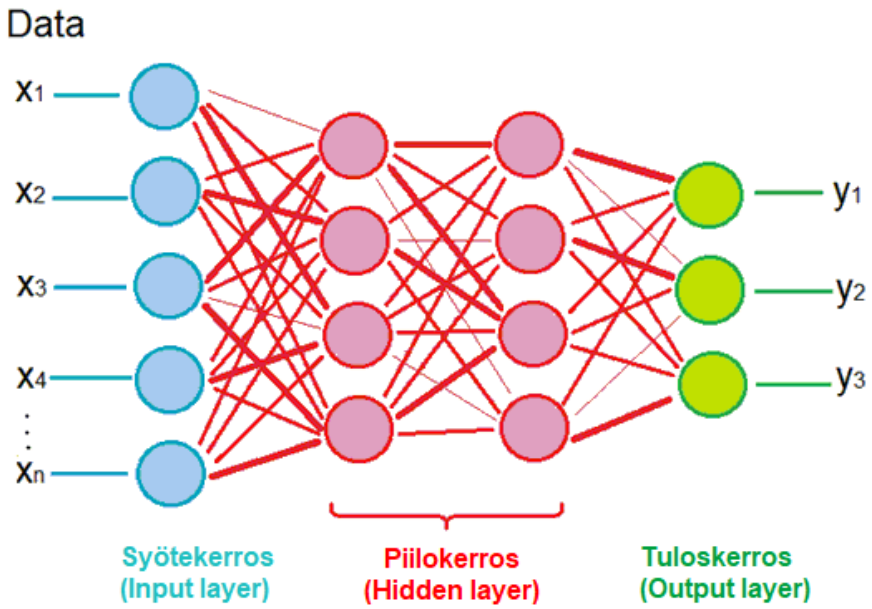


$$s = f(x) = \sum_{i=1}^{n} w_i \cdot x_i + b$$

$$\sigma = \sigma(s) = \frac{1}{1 + e^{-s}}$$

Kuva 8. Yksittäisen neuronin rakenne ja laskentakaava.

Neuroverkot koostuvat useista tällaisista neuroneista, jotka on asetettu samaan kerrokseen toisten neuronien kanssa, ja jotka kommunikoivat seuraavan kerroksen neuronien kanssa, mikä on nähtävissä kuvassa 9. Tällaisia neuronikerroksia voi olla useita, minkä vuoksi neuroverkkojen kerrokset luokitellaan kolmeen erilliseen ryhmään. Syötekerros (input layer) on se kerros, mihin käyttäjä tai ohjelma syöttää datan, kun käyttäjä haluaa kouluttaa tai prosessoida dataa neuroverkon avulla. Tästä kerroksesta tiedot siirtyvät piilokerroksiin (hidden layer), joita yleensä on useita peräkkäisiä kerroksia. Nämä ovat ne kerrokset, joita käyttäjä ei yleensä pysty havainnoimaan mitenkään, eikä pysty tarkastelemaan niiden toimintaa. Näillä kerroksilla tapahtuvat useimmat neuroverkon operaatiot. Neuroverkon viimeistä kerrosta kutsutaan tulostekerrokseksi (output layer), minkä perusteella käyttäjä tai ohjelma tekee päätökset neuroverkon tuloksesta. Yleensä neuroverkkoja käytetään datan luokitteluun, mutta sitä voidaan myös käyttää apuna, kun suodatetaan häiriöaltista dataa ja kun syötetyn datan perusteella luodaan ennusteita. Kuvassa 9 on esitetty nelikerroksinen neuroverkko ja sen rakenne. (Kelleher, 2020, s. 66-68)



Kuva 9. Neuroverkon rakenne.

Oletetaan, että olemme rakentaneet kuvassa esitetyn nelikerroksisen neuroverkon prosessoimaan Twitter viestejä ja neuroverkko pyrkii tunnistamaan, onko viesti sisällöltään positiivinen, negatiivinen vai neutraali. Oletetaan myös, että neuroverkko saa syötteenä Twitter viestin, josta on poistettu yleisimmät täytesanat, kuten esimerkiksi ja, tai, on ja niin edelleen. Nämä jäljelle jääneet tärkeimmät sanat syötetään neuroverkon prosessoitavaksi syötekerrokselle. Tämän jälkeen neuroverkko laskee kerros kerrallaan jokaisen neuronin arvon. Kuvassa on esitetty ohuilla viivoilla heikot yhteydet, missä painon arvo on lähellä nollaa ja paksujen viivojen avulla ne yhteydet missä painon arvo on suuri. Kun kaikki piilotettujen kerrosten neuronien arvot on laskettu, niin lopuksi neuroverkko laskee tulostekerroksen neuronien arvot. Tässä tilanteessa tulostekerroksen arvojen voidaan kuvitella olevan todennäköisyyksiä, missä suurin arvo kuvastaa neuroverkon antamaa ennustetta. Kannatta kuitenkin huomioida, että tämä kyseinen tulos on ainoastaan ennuste, joka saattaa olla virheellinen. Tämä seikka on tärkeää muistaa, kun tarkastellaan ja tutkitaan neuroverkojen toimintaa. (Sugomori & all., 2017.; Warr, 2019.)

4.2 Neuroverkkojen toiminta

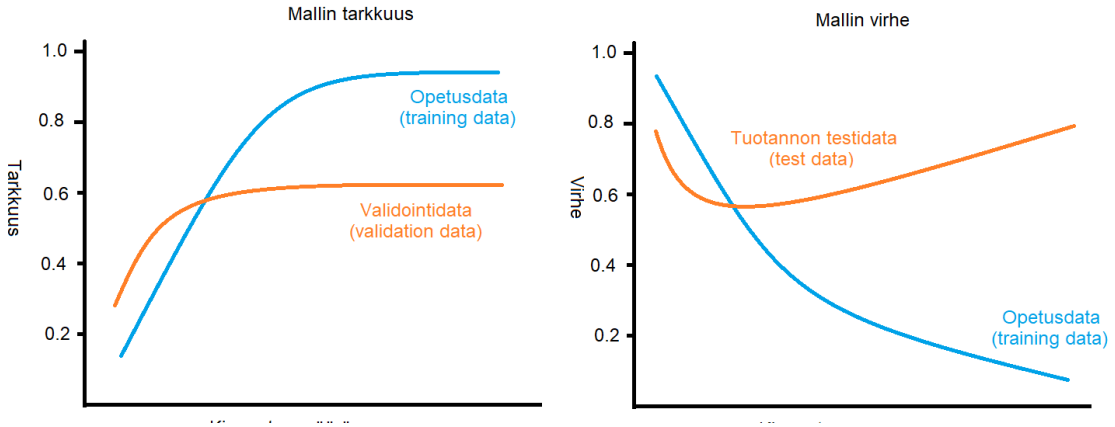
Neuroverkot toimivat siten, että data -analyytikko laatii tietyn levyisen ja syvyisen neuroverkon, minkä koko voi perustua tutkijan kokemukseen tai olettamukseen hyvästä mallista. Tämän jälkeen ohjelma arpoo neuroverkkokerrosten väliset painot (w_i) ja yksittäisten neuronien omat painot (b). Tämän jälkeen tutkija syöttää ohjelmalle koulutusdataa, joiden lopputulos tiedetään. Mikäli ohjelma ennustaa lopputuloksen oikein, solmujen painot kasvavat ja mikäli ohjelma ennustaa väärin, painojen arvoa pienennetään. Tätä neuroverkkojen painojen muuttamismenetelmää kutsutaan vastavirta algoritmiksi (backpropagation

method). Tämä nimitys johtuu siitä, että neuroverkko muuttaa ensin tulostekerroksen painoja virheellisten painojen osalta, mistä se sitten siirtyy muuttamaan painoja kerros kerrallaan kohti sisääntulokerrosta. Tätä kouluttamista jatketaan niin kauan, kunnes neuroverkon tarkkuus ei parane. Yleensä tämä raja -arvo sovitaan ennakolta sopivan korkeaksi, jotta ohjelma ei vahingossa joudu ikuiseen silmukkaan. (Warr, 2019.)

Neuroverkkoja koulutettaessa on otettava myös huomioon, että kyseinen algoritmi kykenee löytämään tarkkuuden suhteen ainoastaan lokaalin maksimin (tai toisin ilmaistuna virheiden suhteen lokaalin minimin), joka riippuu alussa arvottujen painojen arvosta ja koulutusdatasta. Toisin sanoen olemme löytäneet paikallisen parhaan arvon, mutta emme kuitenkaan parasta arvoa tälle neuroverkolle. Tämän takia kannattaa generoida useita neuroverkkoja ja valita niistä se, joka kykenee tuottamaan parhaimman ennusteen syötetystä datasta. Tämä usean neuroverkon generointi ei takaa parhaimman arvon löytämistä, mutta se kuitenkin auttaa parantamaan kehittämäämme neuroverkkoa jonkin verran. Toisaalta meidän ei kuitenkaan kannata etsiä globaalia maksimia, eli parasta mahdollista arvoa, koska tällöin on mahdollista, että olemme onnistuneet ylikouluttamaan neuroverkon. Ylikouluttamisella tarkoitetaan sitä, että malli osaa erittäin hyvin muodostaa ennusteita koulutusdatasta, mutta kun pyydämme neuroverkkoamme muodostamaan ennusteita reaalimaailman datasta, niin ennusteet menevät useimmiten pieleen. (Kathuria, 2023; Keim, 2020)

Nykäsen (2021) Rizzolin (2022) mukaan neuroverkon laatua voidaan kehitysvaiheessa testata vertaamalla opetusdatan tuottamaa mallia validointidataan (validation data), joka on yleensä kooltaan 5–20 % kaikesta kerätystä datasta. Prosenttimäärän valinta riippuu yleensä kerätyn datan määrästä. Mikäli malli osaa ennustaa validointidatan hyvin, niin mallia voidaan pitää potentiaalisesti hyvänä. Tässä kehittämissyklissä on kuitenkin se vaara, että onnistutaan luomaan malli, joka osaa ennustaa opetusdatan ja validointidatan erinomaisesti, mutta reaalimaailmasta poimittua dataa se osaa ennustaa huonosti. Tämän takia suositellaan pistettävän sivuun 5-20 % verran alkuperäistä dataa, eli saman verran kuin laitettiin validointidataa sivuun. Tätä kerättyä dataa kutsutaan tuotannon testausdataksi (test data), eikä sitä saa missään vaiheessa käyttää mallia rakennettaessa ja testattaessa. Kun mallit on saatu rakennettua valmiiksi, niin tällöin voidaan vertailla valmiita malleja tuotannon testausdataan, jonka avulla saadaan todellinen arvio mallin laadusta. Sarang (2020) esitti teoksessaan hyvin, miten koulutuskierrosten määrä saattaa vääristää dataa niin paljon, että kyseinen malli ei pysty ennustamaan oikein reaalimaailman ilmiöitä. Sarangin esittämä kuva oli samankaltainen kuin kuvassa 10. Tässä kuvassa on nähtävillä, kuinka hyvin malli osaa ennustaa koulutusdataa yhä paremmin, mutta se ei kuitenkaan paranna validointidatan tarkkuutta. Oikeanpuoleisessa kuvassa puolestaan on nähtävillä,

miten paljon mallin ennustavuus huononee, kun mallin rakentaminen perustuu liiaksi opetusdataan. Tämä neuroverkkojen opetukseen liittyvä ongelma on otettava huomioon, kun kehitetään malleja. Eli vaikka neuroverkko osaisi selittää äärimmäisen hyvin opetusdataa, niin se ei välttämättä pysty ennustamaan reaalimaailman ilmiöitä. Malleja koulutettaessa on hyvä myös tarkkailla, kuinka paljon validointidatan ennustettavuus paranee. Mikäli validointidatan ennustettavuus ei parane, niin tällöin on syytä lopettaa mallin kouluttaminen ja mahdollisesti pohtia, osaisiko toisenlainen neuroverkko muodostaa parempia ennustuksia.



Kuva 10. Neuroverkkojen koulutusdatan ja tarkkuuden välinen yhteys.

4.3 Neuroverkkojen kehittyminen

Sen jälkeen, kun neuroverkot, sigmoid -funktio ja vastavirta algoritmi keksittiin, ovat neuroverkot kehittyneet yhä monipuolisimmiksi. Nykyään käytössä useita erilaisia aktivointifunktioita, joissa jokaisessa on omat hyvät ja huonot puolensa. Esimerkiksi Choubeyn (2023) on käsitellyt artikkelissaan kuutta yleisintä aktivointifunktiota, sekä esitellyt näiden hyviä ja huonoja puolia. Useat aktivointifunktiot kärsivät häviävän gradientin ongelmasta, kuten esimerkiksi suosittu relu aktivointifunktio. Relu aktivointifunktio voidaan esittää yhtälöllä g (s) = max{0,s} (Brownlee, 2019; Keras, 2023e). Häviävän gradientin oletettiin olevan teoreettinen raja, mutta vuonna 2005 tutkijat havaitsivat sen olevan ainoastaan käytännön este (Kelleher, 2020, s. 133-135). Tämän ongelman takia relu aktivointifunktiosta on jatkokehitetty leaky relu aktivointifunktio, joka voidaan esittää yhtälöllä

$$g(s,h) = \begin{cases} s, \text{kun } s \geq 0 \\ -h \cdot s, \text{kun } s < 0 \end{cases}$$

missä h on jokin ennalta sovittu pieni positiivinen vakio (PyTorch, 2023). Leaky relu ei kärsi häviävän gradientin ongelmasta, mutta sen ongelmana on, ettei se ole laskennallisesti yhtä tehokas kuin alkuperäinen relu aktivointifunktio. Choubeyn artikkelissa myös ilmenee, että alkuperäinen sigmoid aktivointifunktio sopii hyvin luokitteluun, kun taas relu malli sopii parhaiten kuvien luokitteluun tai muiden epälineaaristen ongelmien ratkaisemiseen.

Neuroverkkoihin on kehitetty myös uudenlaisia kerroksia, joiden tarkoituksena on muokata neuroverkoissa kulkevaa dataa halutulla tavalla. Esimerkiksi Kerasissa on käytössä sellainen kerrostyyppi kuin Dropout, jonka asettaa satunnaisesti tiettyjen solmujen lopputuloksen nolliksi ja samalla se estää datan ylisovittamisen. Flatten kerrostyyppiä käytetään yleensä kuvien kanssa, ja kyseisen kerroksen tarkoituksena on muuttaa useampikerroksisen datan yksikerroksiseksi vektoriksi. Kuvien käsittelyä varten on luotu myös muita erilaisia kerrostyyppejä, kuten esimerkiksi Con1D ja Conv2D, joita Ye käy tarkemmin läpi artikkelissansa. (Keras, 2023a; Keras 2023b; Ye, 2020)

On myös kehitetty uudenlaisia vastavirta algoritmeja, joita voidaan käyttää, kun neuroverkkoja koulutetaan. Alkuperäisen gradientin lisäksi on kehitetty esimerkiksi momenttiin ja Adam algoritmin estimaatteihin perustuvia menetelmiä (Keras, 2023c). Lisäksi neuroverkkojen oppimista voidaan arvioida häviöfunktioiden (loss function) avulla. Häviöfunktio antaa arvion siitä, kuinka oikeellisia tuloksia neuroverkko antaa, jonka perusteella vastavirta algoritmi pyrkii korjaamaan neuroverkon painojen arvoja. Suorittamalla tämän saman toimenpiteen tuhansia kertoja peräkkäin, kyseiset algoritmit kykenevät minimoimaan häviöfunktion arvon erittäin pieneksi (Keras, 2023d). Maksutovin (2018) ja Maheshkarin (2022) artikkeleissa tarkastellaan eri vastavirta algoritmeja. Maksutov toteaakin artikkelissaan, ettei ole olemassa mitään yksiselitteistä määritelmää sille, milloin kannattaisi valita tietty vastavirta algoritmi. Hänen kokemuksensa mukaan, neuroverkon asetuksilla ja vastavirta algoritmien parametrien valinnat vaikuttavat enemmän neuroverkon tehokkuuteen ja tarkkuuteen.

5 Tutkimuksen kulku

Tutkimuksen datan keruussa käytettiin apuna Movesensen omilla sivuilla (Suunto, 2021) löytyvää avointa lähdekoodia, minkä avulla Movesensen sensorit voidaan bluetooth teknologian avulla liittää suoraan matkapuhelimeen. Matkapuhelin tallensi kerätyn datan puhelimen omaan SQLite tietokantaan, mistä ne siirrettiin USB:n avulla tietokoneelle käsiteltäväksi. Dataa kerättiin vuoden 2021 kesän ja syksyn aikana tehtäessä erilaisia toimintoja.

Tutkimusasetelma perustuu siihen, että urheilijat ja tavalliset kansalaiset käyttävät sykevyötä yleensä silloin, kun he ovat kuntoilemassa. Sykevyön käyttäjä tai mahdollinen urheiluvalmentaja on yleensä kiinnostunut parantamaan tuloksia. Parhaimmat mittarin henkilön kunnosta tarjoaa sydämen tuottama ECG -käyrä, joka kuitenkin kärsii urheilun ja hengityksen aiheuttamista artefakteista. Tällöin ECG -datan tulkitseminen ja mahdollisella analyysiohjelmalla on vaikeaa tulkita ECG -käyrän rakennetta. Tämän vuoksi tutkimuksen tarkoituksena on tarkastella, kuinka neuroverkko voidaan kouluttaa korjaamaan ECG -käyriä ja kuinka hyviä tuloksia tällainen korjaus tuottaa.

5.1 Tutkimusasetelman muodostaminen

Tutkimusasetelmani teoriana on, että neuroverkko voidaan kouluttaa korjaamaan ECG -käyrää, kun usean sensorin avulla kerätään ECG -dataa liikkuvalta urheilijalta. Jotta neuroverkko saadaan koulutettua, on kyettävä keräämään tarpeeksi paljon luotettavaa dataa sydämen toiminnasta. Yksittäinen sykevyö ei monesti kykene keräämään tarpeeksi paljon dataa sydämen toiminnasta, minkä vuoksi koulutettava ECG -data olisi hyvä kerätä kolmen erillisen sensorin avulla. Jotta näitä kolmea kerättyä ECG -dataa voitaisiin verrata keskenään, on sensorit kyettävä synkronoimaan keskenään. Lisäksi on huomioitava, että sensorien sijoittelu vaikuttaa myös tallennetun ECG -käyrän rakenteeseen. He, Cliffordin & Tarassenkon (2006, sivu 107) perusteella parhaimmiksi sijoituspaikoiksi valikoitui V3, V4 ja V6, jotka ovat nähtävissä myös kuvassa 3, sivulla 12. Näistä V3 ja V4 sijaitsevat suoraan sydämen päällä, mutta kolmatta sensoria on erittäin vaikeaa saada mahtumaan samalle alueelle. Tästä syystä vasen kylki oli mielestäni paras paikka kolmannen sensorin sijoittamiselle. Tämä sensorien asettelu osoittautui toimivaksi ja sykevyöt eivät asettuneet liiaksi toistensa päälle. Näiden kerättyjen ECG -datojen pohjalta on tarkoituksena luoda ECG -käyrän malli, mitä käytetään pohjana koulutettaessa neuroverkko korjaamaan ECG -käyrän artefakteja.

5.2 Datan keruuseen liittyvät haasteet

Movesensen omilla sivuilla (Suunto, 2021) löytyy avoin lähdekoodi, minkä avulla heidän tuottamat sensorit voidaan liittää matkapuhelimeen. Tämän lähdekoodin heikkoutena oli

kuitenkin se, että tiedot tallennettiin matkapuhelimen CSV -tiedostoon. Mikäli matkapuhelin jossain vaiheessa kaatui, katosivat samalla tiedostoon tallennetut tiedot. Toinen ongelma, joka ilmeni vuoden kesällä 2021, oli tiukentuneet turvavaatimukset tiedostojen tallennukselle. Tämän takia alkuperäistä ohjelmaa oli muokattava siten, että tiedot tallennettiin matkapuhelimen SQLite -tietokantaan, jolloin ohjelman mahdollinen kaatuminen ei hävittäisi jo kerättyjä tietoja.

Useamman sensorin käyttöön liittyy myös se ongelma, että monen sensorin kytkeminen matkapuhelimeen saattaa kaataa sen. Lisäksi alkuperäinen lähdekoodi ei mahdollistanut kuin kahden sensorin liittämisen matkapuhelimeen. Tämän takia valmiiseen ohjelmaan oli ohjelmoitava oma osio, johon voitiin liittää kolme sensoria samanaikaisesti. Jotta vältyttäisiin matkapuhelimen kaatumiselta, sensorit oli ohjelmoitava siten, että ainoastaan yksi sensori kerää liikedataa, gyroskooppidataa ja ECG -dataa. Kaksi muuta sensoria ohjelmoitiin keräämään ainoastaan ECG -dataa. Tämä ratkaisu osoittautui toimivaksi, eikä sovellus kaatunut sensoreiden määrän vuoksi.

Sovellusta ohjelmoitaessa ilmeni, että Movesensen tärkein whiteboard -kirjasto on suunniteltu aiemmille Android -järjestelmille. Uusimmat Android Studiot eivät automaattisesti tue näitä vanhoja versioita, vaan ohjelman kirjastoihin on tehtävä pieniä muutoksia ja tuki on asetettava erikseen päälle Android Studio ympäristössä.

Vuoden 2021 aikana turva -asetukset tiukentuivat myös foreground- ja background sovellusten osalta sen verran, että jotkin Android versiot pyrkivät hanakasti sammuttamaan kyseisiä palveluja käyttävät sovellukset. Silloin Android järjestelmä lopettaa kyseisten palvelujen toiminnan, mikäli puhelin siirtyy “lepotilaan”. Jotkin ohjelmat pyrkivät kiertämään tämän satunnaissynkronoinnin avulla, minkä lisäksi säännöllinen yhteydenotto sovelluksen palvelimelle auttaa ylläpitämään nämä palvelut toiminnassa.

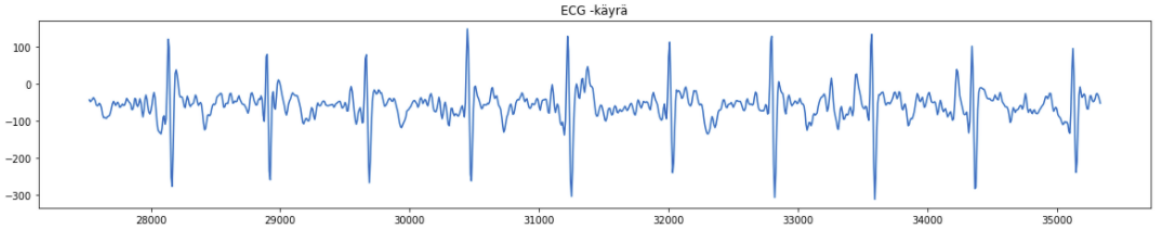
Näiden haasteiden vuoksi en kolmesta sensorista ei ehditty keräämään paljon dataa, mutta kuitenkin sen verran että sen avulla kykeni kouluttamaan neuroverkon. Koulutuksessa käytetty tietokanta sisälsi noin 9,7 miljoonaa riviä dataa, jotka oli kerätty kaikista kolmesta sensorista. Pythonin Neurokit2 -kirjaston mukaan, kukin sensori oli ehtinyt keräämään noin 7 600 R -piikkiä neuroverkon kouluttamista varten.

Halusin kuitenkin saada kouluttamista varten lisää ECG dataa, jolloin eräs mahdollinen ratkaisu on käyttää apuna jotakin olemassa olevaa kirjastoa. Kirjaston löytämisen tekee kuitenkin hankalaksi se, että useimmat ECG -käyrät on poimittu lepotilassa olevilta potilailta tai sitten liikedataa ei olla kerätty potilailta. Löysin kuitenkin Physionetistä aineiston

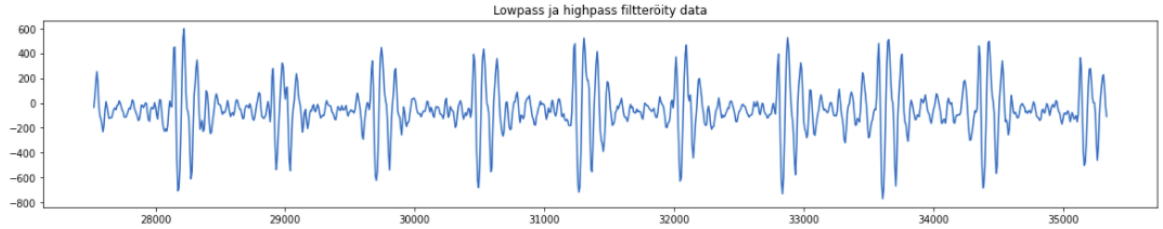
(PhysioNet, 2015; Goldberg & all. 2000), jossa oli kerätty 13 eri potilaalta artefakteja sisältävää dataa, jonka keräämisessä käytettiin apuna viittä erilaista sensoria (Vollmer & all., 2022; Goldbeerger & all., 2000). Näiden tietojen avulla kykenin luomaan neuroverkon ja tarkastelemaan sen tuottamia tuloksia.

5.3 ECG algoritmien luonti

Monet ECG -algoritmit perustuvat Pan-Tompkins algoritmiin, joka pyrkii tunnistamaan ECG -käyrän R -piikit. Kyseisessä algoritmissa prosessoidaan ECG -käyrä lowpass ja highpass suodattimien avulla ja niiden tarkoituksena on korostaa R -piikin taajuusalueeseen 5–15 Hz kuuluvia piikkejä. Samanaikaisesti kyseiset suodattimet eliminoivat muun taajuisia piikkejä datasta.

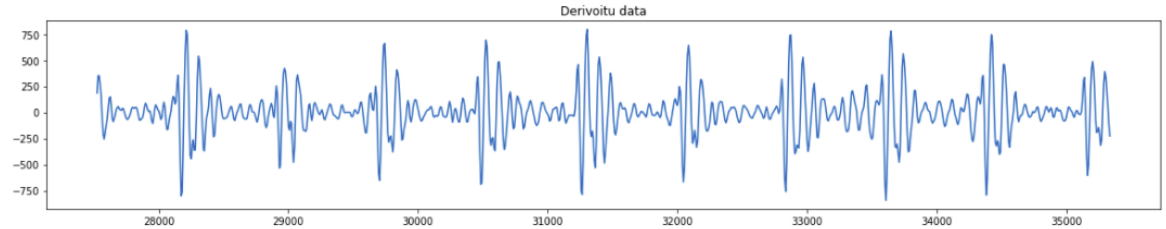


Kuva 11. Raaka-data



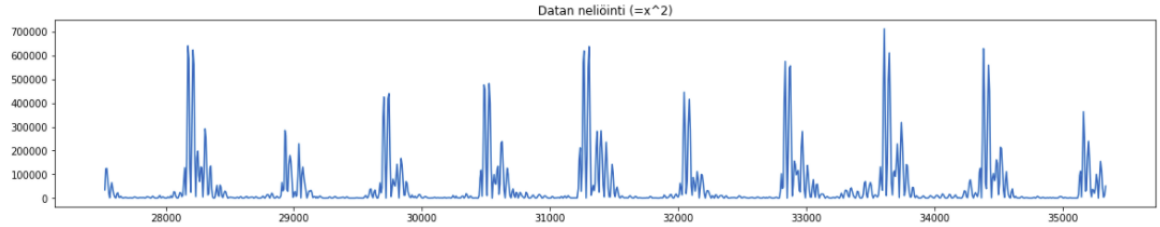
Kuva 12. Filtteröity data

Datassa saattaa esiintyä joskus portaan tapaisia askelmia, mikä häiritsee algoritmien tulkitsemista. Tämän vuoksi data on derivoitava, jotta data saadaan korjattua halutunlaiseksi. Lisäksi datapisteet, joiden välillä on pienet erot, pienenevät entisestään, jolloin R -piikki näkyy selvemmin kerätyssä datassa.

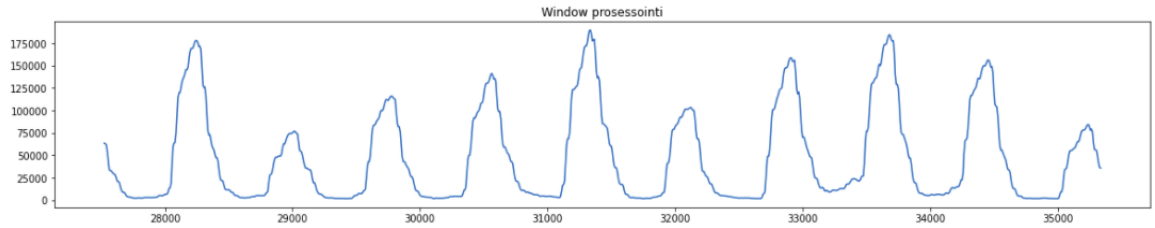


Kuva 13. Derivoitu data

Lopuksi data neliöidään, jolloin saadaan paremmin korostettua R -piikin arvoja ja lähellä nollaa olevat arvot pienevät entisestään. Lopuksi jokaisen datan piste integroidaan ennalta sovitun aikavälin yli, jolloin muodostuu tasainen kumpumaisema.



Kuva 14. Datan neliöinti



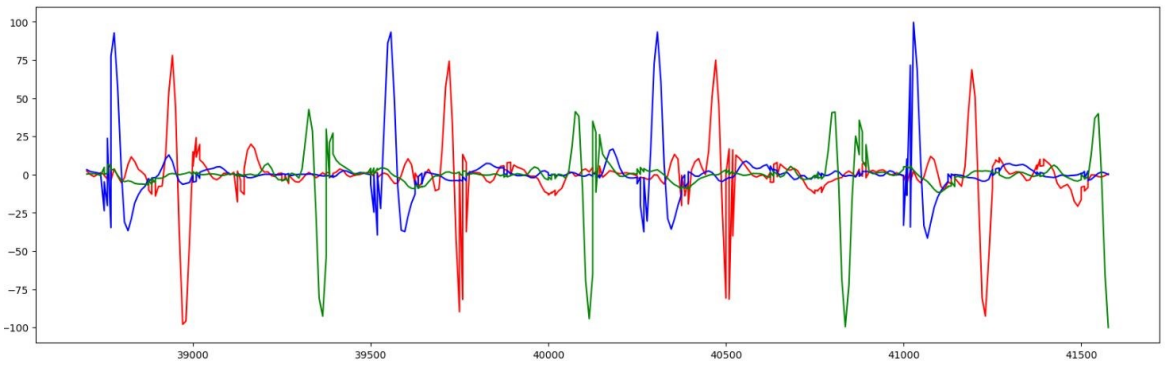
Kuva 15. Data integraatio sovitun aikavälin yli (moving window integration)

Kun nämä kumpumaiseman arvon skaalataan välille [0,1], jolloin voidaan asettaa raja, jonka yläpuolella tarkasteltavien kumpujen huippujen on oltava. Tällöin R -piikkien etsintä rajoitetaan näihin rajan yltäviin kumpuihin.

Tutkimuksen aikana kävi ilmi, että useimmat R -piikkien etsimisalgoritmit eivät löydä kovinkaan hyvin R -piikkejä raaka -datasta, vaan ECG data on ensin normalisoitava, mikä puolestaan voidaan tehdä useilla eri tavoilla. Normalisoinnissa päädyin käyttämään liikkuvalla aikavälillä poimittuja mediaanien arvoja korjatakseni datan x -akselin suuntaiseksi ja siirtääkseni havainnot mahdollisimman lähelle nollaa. Päädyin käyttämään tässä tapauksessa mediaania, koska se antaa luotettavamman arvon kuin keskiarvo, minkä lisäksi tämä auttoi pääsemään eroon ECG:n perustason vaelluksesta. Tämän jälkeen käytin liikkuvaa skaalausta siten, että liikkuvalla aikavälillä olevat datan arvot asettuivat [-100, 100] välille. Liitteessä yksi on nähtävissä, miten tämä datan normalisointi suoritettiin. Tutkimuksessani ilmeni, että tämän normalisoinnin seurauksena useimmat ECG -algoritmit kykenivät löytämään R -piikit tehokkaimmin. Näiden R -piikkien löytäminen oli tärkeää, kun halusin suorittaa lopullisen synkronoinnin kolmen eri sensorin kanssa, josta kerrotaan seuraavassa luvussa.

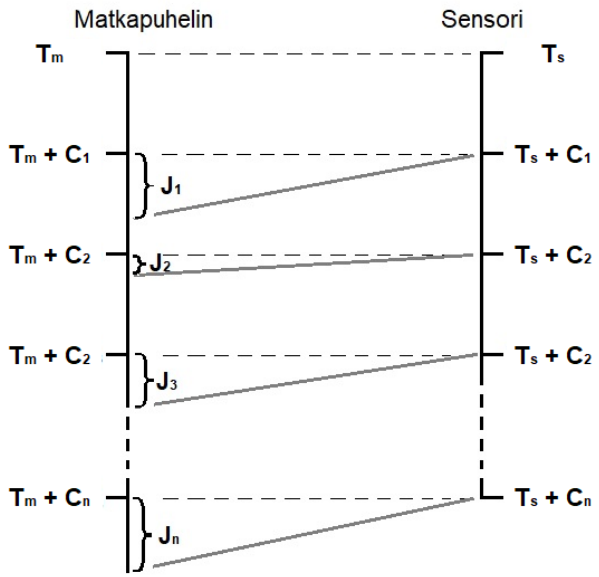
5.4 Kerätyn aineiston synkronointi

Ensimmäinen ongelma aineiston käsittelyssä on se, ettei sensoreita ole kalibroitu tiettyyn hetkeen, koska Movesense sivuilta ei löydy siihen selkeää ohjeistusta. Ensimmäinen ajatus datan synkronoinnille on käyttää synkronoinnissa sensorien aikaleimoja, missä kaikista aikaleimauksista vähennetään ensimmäisen aikaleiman arvo. Tällöin kaikkien kolmen sensorin aikaleima lähtee liikkeelle nollasta ja voidaan ajatella, että sensorien ajanhetket ovat samoja. Näin tehtäessä saadaan ECG -käyrästä muodostettua kuvan 16 kaltainen esitys. Tästä kuvasta voidaan suoraan havaita, että sensorien keräämät datat eivät ole synkronoituja keskenään, eli on kehitettävä vieläkin parempi keino.



Kuva 16. Mediaanin avulla korjattu ja skaalattu raaka ECG data

Toinen keino ratkaista sensoreiden kalibrointiongelma on kerätä kaksi erillistä aikaleimaa, joista toinen aikaleima poimitaan sensorista ja toinen matkapuhelimen omasta järjestelmästä. Tällöin sensorin lähettämien tietojen ajanhetket ja matkapuhelimen vastaanottaman tietojen ajanhetket voidaan esittää kuvan 17 avulla.



Kuva 17. Matkapuhelimen ja sensorien välinen ajallinen kommunikointi. Huomaa että J_i:t eivät ole yhtä isoja, vaan ne kuvaavat satunnaista viivettä, joka liittyy tiedon vastaanottamiseen.

Koska matkapuhelinta ja sensoria ei olla synkronoitu keskenään, niin sensorin tallennusajankohdan alkua merkitään T_s:llä, mikä vastaa matkapuhelimessa ajankohtaa T_m. Sensori kerää ympäristöstänsä tietoja C_1, C_2, …, C_n välisen ajan, minkä jälkeen se lähettää kerätyt tiedot matkapuhelimelle tallennettaviksi Bluetooth yhteyden kautta. Koska kulunut aika on molemmilla laitteille sama, niin sensorin ajankohdat $T_s + C_k$ vastaavat matkapuhelimen ajankohtia $T_s + C_k$, missä k=1, 2, …, n. Käytännössä sensori lähettää tietoja säännöllisin väliajoin, mutta tätä olettamusta ei tarvita laskennassa. Ainoa epäsäännöllisyys matkapuhelimen tallennusajankohdassa johtuu tietojen siirtymisestä Bluetoothin kautta, mihin liittyvää satunnaista viivettä merkitään J_k:lla, missä k=1, 2,…, n. Toisin ilmaistuna, kun matkapuhelin tallentaa sensorin tiedon, niin matkapuhelin merkitsee tiedon tallennusajankohdaksi $T_1 + C_k + J_k$, missä k=1, 2, …, n. Teoriassa tämä viive saattaa olla niin iso, että matkapuhelin vastaanottaa myöhemmin lähetetyn viestin, ennen kuin aikaisemmin lähetetty viesti tallentuu järjestelmään. Koska tietoja tallennetaan jatkuvalla syötöllä yli tunnin ajan, niin tämän ongelman ei pitäisi tuottaa ongelmia, koska järjestelmä onnistuu tänä aikana keräämään tarpeeksi paljon vertailtavia arvoja.

Kun tarkastellaan matkapuhelimen tallennushetken ja sensorin tallennushetken erotusta, niin voidaan havaita, että

$$E_1 = (T_m + C_1 + J_1) - (T_s + C_1) = T_m - T_s + J_1$$
$$E_2 = (T_m + C_2 + J_2) - (T_s + C_2) = T_m - T_s + J_2$$
$$E_3 = (T_m + C_3 + J_3) - (T_s + C_3) = T_m - T_s + J_3$$
$$\dots$$

$$E_n = (T_m + C_n + J_n) - (T_s + C_n) = T_m - T_s + J_n.$$

Silloin voidaan laskea pienin mahdollinen siirtymä koko yhtälöllä

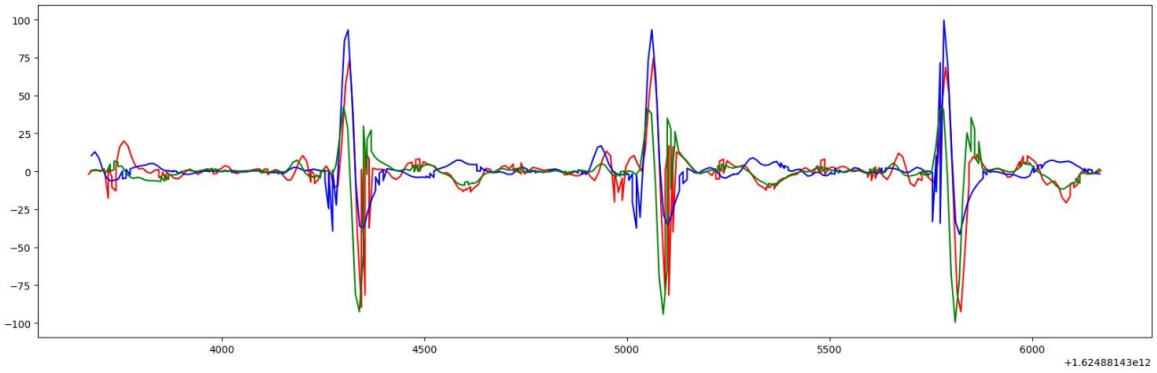
$$E = \min_{k=1,2,..,n} E_k - T_m + T_s$$
$$= \min_{k=1,2,...,n} J_k.$$

Kun sensorin tallentamasta aikamerkinnöistä vähennetään tallennuksen alkuhetki S1, jolloin sensorin tallennusajakohdiksi saadaan arvot C_1, C_2, …, C_n. Kun tämä arvo lisätään matkapuhelimen tallennusajakohdan arvoon ja vähennetään tietojen siirtymään kulunut aika E, niin tällöin sensoreiden korjattu ajankohta lasketaan yhtälöllä

$$t_k = T_m + [(T_s + C_k) - T_s] - E$$
$$= T_m + C_k - E,$$

missä k=1, 2, …, n. Tästä arvosta voitaisiin vähentää arvo C_1, mutta koska sensori lähettää tiedot matkapuhelimelle säännöllisin väliajoin, niin tätä vähennystä ei tarvitse tehdä. Tämä yhtälö antaa melko tarkan arvion siitä ajankohdasta, jolloin sensoritieto on kerätty. Kun tätä yhtälöä sovelletaan kaikkiin sensoreihin ja niiden keräämiin tietoihin, niin kaikkien kerättyjen tietojen pitäisi olla synkronoituja keskenään, mikä helpottaa aineiston prosessointia ja analysointia.

Kun data synkronoidaan tämän menetelmän avulla, niin voidaan havaita, että bluetoothin viive on alimmillaan noin 15–20 millisekunnin luokkaa. Tästä korjatusta aikaleimadatasta voidaan luoda seuraavanlainen kuva.



Kuva 18. Aika- ja mediaanikorjattu ECG -data, joka on vielä skaalattu.

Kuvasta on nähtävissä, että tämä kahden aikaleiman järjestelmä synkronoi datan lähes kohdalleen. Kuvaajaa katsoessa kyetään havaitsemaan, että vihreän sensorin R -piikki ilmenee hieman aikaisemmin kuin punaisen sensorin R -piikki. Toisin sanoen data saatiin synkronoitua melko tarkasti, mutta ei kuitenkaan täysin.

Kolmas synkronointiin liittyvä ongelma on sensorien sijoittelu keholla. Eriävät etäisyydet sydämestä ja henkilön fyysinen rakenne, kuten rasvakudoksen sijainti sekä määrä aiheuttavat kerätyssä datassa millisekuntien eroja, jotka on myös kyettävä korjaamaan, jotta

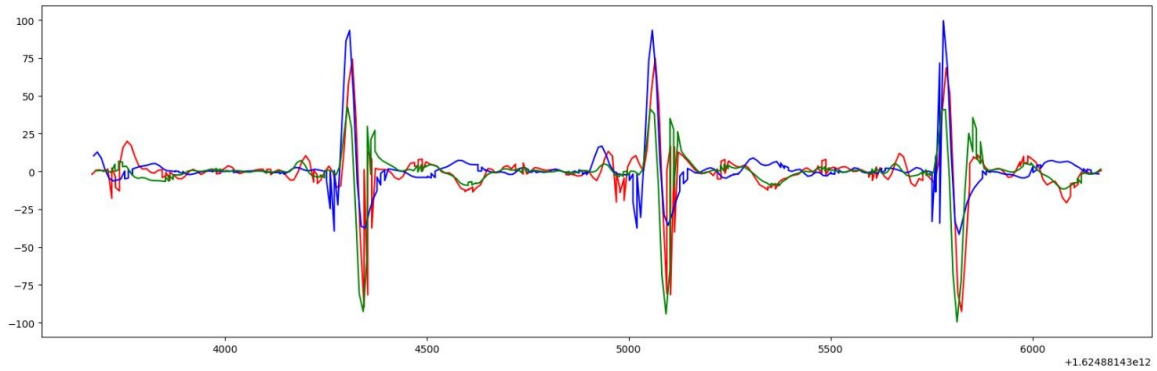
sensorien keräämä data saataisiin synkronoitua. Tämä korjaus onnistuu, kun kaikista kolmesta ECG -käyrästä etsitään R -piikit Pan-Tompkins algoritmin avulla, josta löytyy enemmän tietoa seuraavassa luvussa. Näistä kolmen sensorin R -piikeistä poimitaan ne, joiden välinen etäisyys toisten sensorien R -piikeistä on alle 20 millisekuntia. Kun ensimmäisen sensorin aika oletetaan oikeaksi, niin silloin meidän tarvitsee ainoastaan korjata kahden muun sensorin aika vastaamaan ensimmäisen sensorin aikaa. Selkeyden vuoksi ensimmäisen sensorin R -piikkejä voidaan merkitä Q_i, missä saa arvot i=1, 2, … m, ja kahden muun sensorin R -piikkejä merkitään $R_{2,i}$ ja $R_{3,i}$, missä i=1, 2, ..., m. Tällöin voidaan laskea datan korjausarvo laskemalla R -piikkien välisen etäisyyden keskiarvo yhtälöllä

$$d_2 = \frac{1}{m} \sum_{i=1,2,\dots,m} (R_{2,i} - Q_i) \text{ ja}$$

$$d_3 = \frac{1}{m} \sum_{i=1,2,\dots,m} (R_{3,i} - Q_i).$$

Nämä lasketut d_i -arvot on vähennettävä toisen ja kolmannen sensoreiden aikasynkronoiduista arvoista, jolloin R -piikit saadaan synkronoitua ensimmäisen sensorin aikajanan kanssa.

Korjattaessa kehon aiheuttamaa synkronointiongelmaa, voidaan havaita kehon rakenteen aiheuttavan noin 0–10 millisekunnin viiveen. Kun tämä korjaus tehdään dataan, saadaan muodostettua seuraavanlainen kuva.



Kuva 19. Täysin synkronoitu ja skaalattu ECG -data

Kuten kuvasta nähdään, niin ero näiden kahden synkronoidun datan kesken ei ole suuri, mutta se auttaa kuitenkin muodostamaan kuvan tutkittavan henkilön oikeasta ECG:stä. Liitteessä kaksi on nähtävissä, miten tämä datan synkronointi suoritettiin käytännössä.

5.5 ECG datan korjaus

Kun sensoreiden keräämä ECG -käyrät on saatu synkronoitua keskenänsä, niin tämän jälkeen on kyettävä muodostamaan niistä ECG:n malli, jota käytetään apuna koulutettaessa

neuroverkkoa. Kuten täysin synkronoidusta ECG -käyrästä voidaan havaita, niin sinisen sensorin T -aalto saa positiivisia arvoja, kun taas kaksi muuta sensoria saavat negatiivisia arvoja. Tämän vuoksi käsitellään datan positiivisuutta ja negatiivisuutta erikseen jokaisessa datapisteessä ja prosessoimme laskennassa datan itseisarvoja. Kun tällöin ECG -käyrän yksittäistä pistettä merkitään di:llä, voidaan muodostaa funktiot

$$a(\,d_i\,) = |d_i| = \begin{cases} d_i, \text{kun } d_i \geq 0 \\ -d_i, \text{kun } d_i < 0 \end{cases}$$

ja

$$\sigma(\,d_i\,) = \begin{cases} 1, \text{kun } d_i \geq 0 \\ -1, \text{kun } d_i < 0. \end{cases}$$

Toisin ilmaistuna kaikki datapisteet voidaan esittää yhtälöllä $d_i = \sigma(\,d_i\,) * a(\,d_i\,)$. Tällöin voidaan laskea ECG:n painotettu keskiarvo. Painotetun keskiarvon laskennassa on kuitenkin se ongelma, että on päätettävä minkä sensorin arvoja painotetaan tämän painotettu keskiarvon laskennassa. Ratkaisuni onkin ensin laskea sensoridatojen itseisarvo yksittäisissä pisteissä, minkä jälkeen lasketaan painotettu keskiarvo niiden itseisarvojen mediaanien suhteen. Tämän jälkeen ainoaksi ongelmaksi muodostuu, onko kyseinen datan piste positiivinen vai negatiivinen. Ratkaisuksi saadaan, kun otetaan avuksi funktio

$$\bar{\sigma}(d_1, d_2, d_3) = \begin{cases} 1, \text{kun } \sigma(d_i) = \sigma(d_j) = 1, \text{joillakin i}\neq\text{j} \\ -1, \text{muulloin.} \end{cases}$$

Toisin sanoen datan piste saa positiivisen arvon, jos kahden sensorin arvot ovat positiivisia kyseisessä pisteessä ja datan piste on negatiivinen, jos kahden sensorin arvot ovat negatiiviset kyseisessä pisteessä.

Kun yksittäisen sensorin datapisteistä käytetään merkintää $d_{(s,i)}$, missä s =1,2,3, voidaan luoda sellaiset funktiot kuin

$$m(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) = \min_{k=1,2,3}\{a(d_{(k,i)})\},$$

$$M(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) = \max_{k=1,2,3}\{a(d_{(k,i)})\}$$

ja

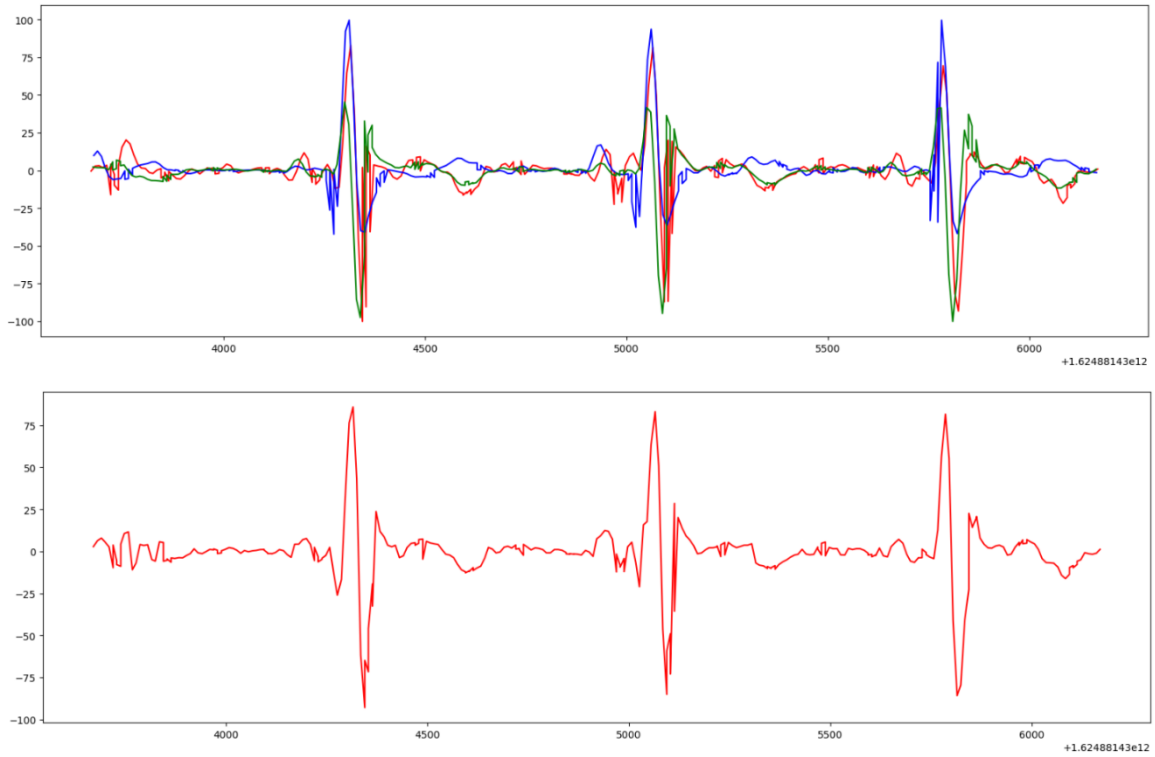
$$G(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) = \operatorname{median}_{k=1,2,3}\{a(d_{(k,i)})\}.$$

Silloin painotetun keskiarvon yhtälö voidaan esittää yhtälöllä

$$P(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) = \frac{1}{10} \cdot \bar{\sigma}(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) \cdot$$
$$[4 \cdot G(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) +$$
$$m(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)}) +$$
$$5 * M(\,d_{(1,i)}\,, d_{(2,i)}, d_{(3,i)})]\,.$$

Toisin sanoen tämän painotetun funktion tarkoituksena on pyrkiä painottamaan datan oikeellisuutta niin, että mediaani saa suurimman painoarvon, minkä jälkeen dataa painotetaan maksimiarvolla. Liitteessä neljä näkee, miten tämä painotettu keskiarvo lasketaan käytännössä. Sen lisäksi tarkoituksena on korjata ECG:n aikamerkinnät ensimmäisen

sensorin aikamerkintöjen mukaisiksi. Tällöin datasta saadaan muodostettua seuraavanlainen kuvaus.



Kuva 20. Eri sensoreiden datasta laskettu mediaani

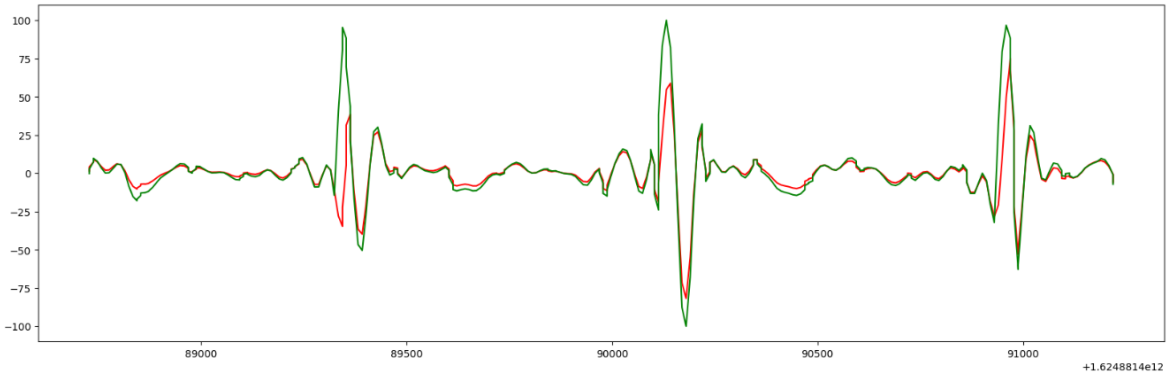
Vaikka malli näyttääkin hyvältä tässä kohtaa, niin joidenkin datapisteiden kohdalla ECG -käyrien piikit aiheuttivat pieniä ongelmia, tai R -piikki oli liian matala. Tämän vuoksi poimin jokaisen R -piikin koordinaatin ja valitsin korkeimman R -piikin näistä kerätyistä ECG -käyristä, minkä perusteella R -piikit voidaan tuoda paremmin esille, kun käytetään apuna sin -funktiota. Kun poimitaan ECG -käyrien korkein R -piikin arvo yhtälöllä

$$M_R = \max\{d_{(1,R)}, d_{(2,R)}, d_{(3,R)}\},$$

sekä valitaan ECG -käyrän koordinaatit R:n molemmilta puolilta ennalta sovitun säännön mukaisesti, voidaan näitä poimittuja koordinaatteja merkitä (x_0, y_0), (x_R,y_R) ja (x_1, y_2). Tässä on otettava huomioon se, että y:n arvot tässä eivät vastaa tiettyä ECG -käyrän arvoa, vaan se on kolmen ECG -käyrän mediaaniarvo kyseisessä pisteessä. Tällöin yhtälöllä

$$y(t) = \begin{cases} y_0 + \sin(\,(\text{t-}x_0)/(x_R\text{-}x_0)\,\cdot\frac{\pi}{2})\,\cdot(y_R - y_1), \text{kun } x_0 \le t \le x_R \\ y_R + [1 - \sin(\text{-}\frac{3\pi}{2} + (\text{t-}x_R)/(x_1\text{-}x_R)\,\cdot\frac{\pi}{2})]\,\cdot(y_R - y_1), \text{kun } x_R \le t \le x_1 \end{cases}$$

saadaan luotua parannettu ja luonnollisempi R -piikki, joka on nähtävissä myös alla olevassa kuvassa.



Kuva 21. Paranneltu ECG -käyrän R -piikki

Punainen käyrä on alkuperäinen painotettuun keskiarvoon perustuva ECG -käyrä ja vihreä käyrä puolestaan on paranneltu versio, jonka avulla R:n piikit saadaan luonnollisemmiksi. Liitteessä kolme on nähtävissä, miten datan siivous ja R -piikkien korostaminen on suoritettu.

5.6 Neuroverkon luominen

Ennen kuin neuroverkkoa ryhdyttiin kouluttamaan, oli sensorien keräämä kiihtyvyysanturin keräämä data myös synkronoitava. Koska kiihtyvyysanturien aikaleima oli aina kytköksissä yhden sensorin aikaleimaan, niin samalla kun korjasin ECG -käyrän synkronointia, korjasin saman aikaisesti kiihtyvyysanturinaikaleimoja. Toinen kiihtyvyysantureihin liittyvä ongelma oli se, että ECG -anturit ja kiihtyvyysanturit keräsivät dataa eri tahdissa. Movesensen sensoreissa ei löytynyt sellaista vaihtoehtoa, että kiihtyvyysantureiden dataa olisi voinut kerätä samassa tahdissa. Tämän vuoksi käytin neliöspliniä apuna estimoidakseni kiihtyvyyteen liittyvät arvot niissä kohdissa, joissa ECG -data oli kerätty.

Neuroverkkojen luomisessa käytettiin apuna Kerasin kirjastoja (Chollet & all., 2015). Tarkoituksenani on aineistoin pienuuden vuoksi rakentaa mahdollisimman pieni ja tehokas neuroverkko prosessoimaan ECG -käyriin liittyviä artefakteja. Koska aineistoni koko ei ollut tarpeeksi suuri, hyödynsin Physionetin aineistoa, jonka avulla kykenin muodostamaan pohjan neuroverkolleni (PhysioNet, 2015; Vollmer & all., 2022; Goldbeerger & all., 2000). Kyseinen aineisto oli saatavilla jo synkronoidussa muodossa, mutta ECG -käyrä piti normalisoida, ennen kuin sitä voitiin käyttää neuroverkon kouluttamisessa. Käyttämästäni normalisointimenetelmästä kerroin jo kappaleessa ECG algoritmien luonti, mikä on myös nähtävissä liitteessä yksi.

Kouluttaessani neuroverkkoa kerätyn datan ja Physionetin aineiston avulla, jouduin pohtimaan millaisilla mittareilla arvioin malleja keskenään. Päädyin siihen tulokseen, että paras neuroverkko on sellainen, joka pääsee parhaimpaan tarkkuuteen mahdollisimman vähillä

koulutuskierroksilla ja jonka virheiden neliösumma on pienin mahdollinen arvo. Tästä virheen neliösummasta lasketaan usein myös neliöjuuri, mutta käytännössä tämä ei vaikuta mallien vertailuun.

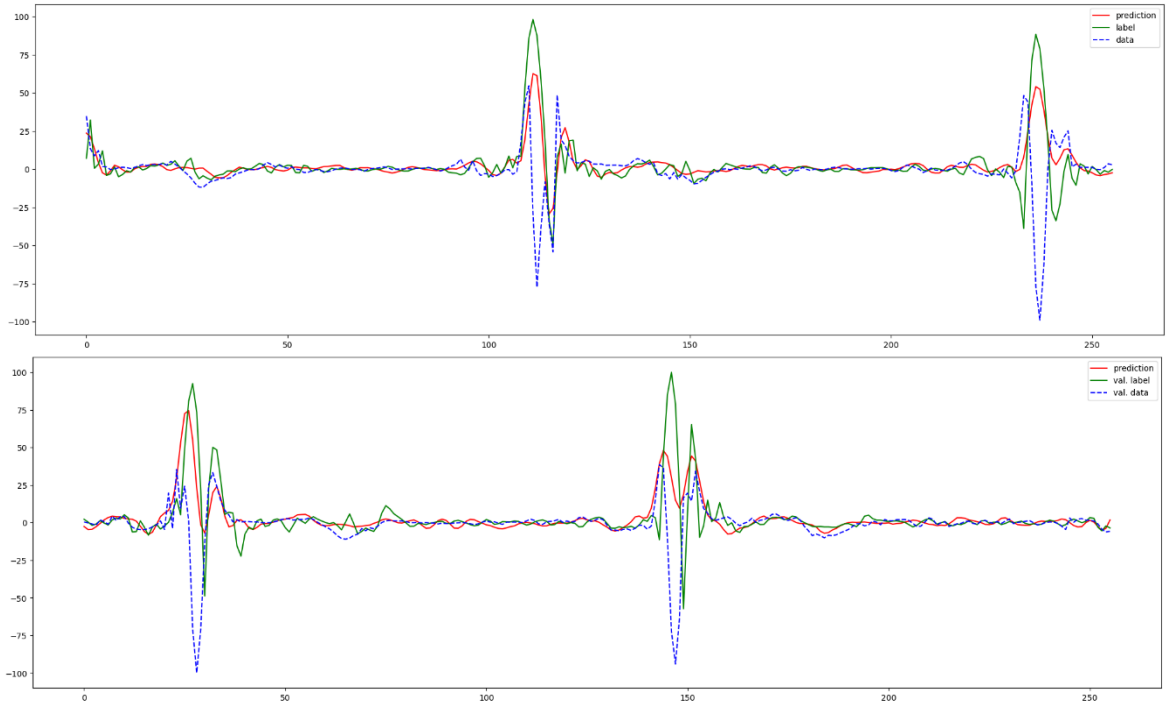
$$\text{Oikea arvo } L_i, \text{ kun } i = 1, 2, \dots, n$$
$$\text{Ennuste } E_k, \text{ kun } k = 1, 2, \dots, n$$
$$\text{Neliösumma } S^2 = \sum_{j=1}^{n} \left(L_j - E_j\right)^2$$

Kuva 22. Neliösumman kaava ennustemallin virheelle

ECG -aineiston prosessointia varten ei kuitenkaan ollut olemassa valmista ohjelmistoa, joka olisi osannut jakaa ECG -käyrän validointi ja koulutusdataan satunnaisesti, minkä vuoksi loin itse tällaisen ohjelman. Valitsin poimittavan datan kooksi 256 datapistettä, joka vastaa yhden sekunnin aikana koottua ECG -käyrästä ja liikeantureista koottua dataa. Neuroverkkojen kouluttamista varten normalisoin datan. Jotta aineisto olisi vertailukelpoista ja se ei vääristyisi, niin validointidata, testidata ja koulutusdata eivät saa sisältää yhteisiä datapisteitä. Liitteessä kuusi on esitetty algoritmi, joka arpoo ja kerää ECG datasta validointi- ja testidatan, sekä samanaikaisesti se korvamerkitsee nämä kohdat käyttökieltoon. Koulutusaineisto valittiin myös täysin satunnaisesti ja siihen sovellettiin ainoastaan sitä ehtoa, ettei sillä saa olla yhteisiä pisteitä validointi- ja testidatan kanssa.
Eri mallien vertailussa käytin testidataa, mitä ei käytetty koulutettaessa malleja. Malleja vertailtaessa kyettiin havaitsemaan, että jotkin tietyt Kerasissa käytetyt kerrostyypit itseasiassa huononsivat malleja ja niiden tarkkuusastetta. Esimerkiksi Dropout -kerros huononsi neuroverkon tarkkuutta ja lisäsi tarvittavien opetuskierrosten lukumäärää. Tämän lisäksi malleja rakennettaessa havaittiin, että leaky reluun perustuva neuroverkko antaa kaikista parhaimmat tulokset pienemmillä kierrosten lukumäärillä. Samalla erilaisia malleja kokeillessa tuli ilmi, että myös liian suuri neuroverkon kerrosten lukumäärä voi heikentää neuroverkkojen tarkkuutta. Parhaimmaksi malliksi osoittautui kymmenkerroksisen neuroverkko, joka on esitetty liitteessä seitsemän. Kun paras mahdollinen neuroverkkomalli oli löydetty, luotiin tämän jälkeen kaksikymmentä erillistä neuroverkkoa kyseisestä mallista ja niistä valittiin paras mahdollinen neuroverkko perusmalliksi jatkokoulutusta varten. Vertailussa käytin mittarina virheen neliösummien arvoja, minkä perusteella pienimmän virheen omaava malli poimittiin tutkimukseen.

Seuraavassa kuvassa nähdään, kuinka hyvin poimittu neuroverkko kykenee mallintamaan koulutusdataa ja sen alapuolessa olevassa kuvassa puolestaan havainnoidaan, kuinka

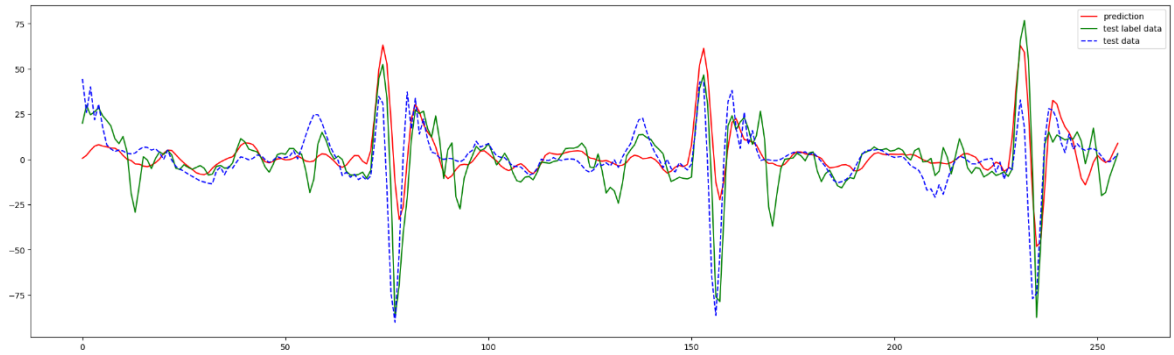
hyvin aineisto pystyy mallintamaan validointidataa. Kuvassa on selkeästi nähtävissä neuroverkon erikoispiirre, eli neuroverkko kykenee mallintamaan erinomaisesti koulutusdataa, mutta validointidatassa on nähtävissä jonkin verran häiriötä.



Kuva 23. Yksi satunnaisesti valittu kohta, missä neuroverkko on luonut ennusteen koulutusdatasta ja validointidatasta.

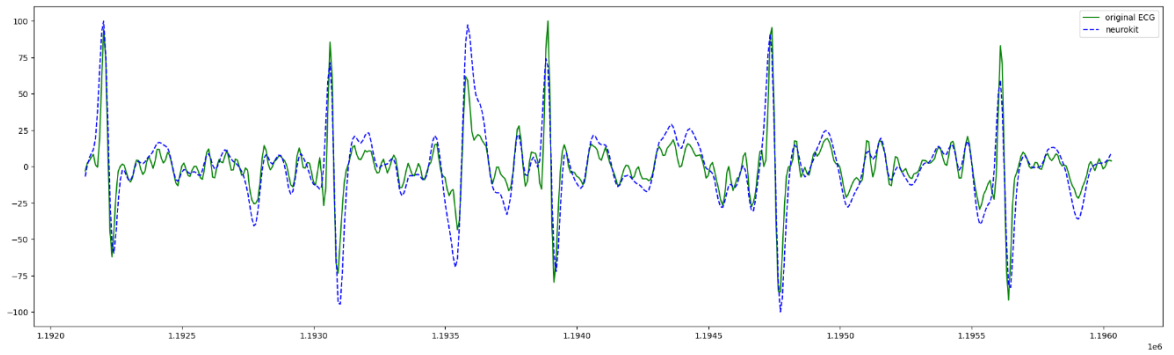
5.7 Aineiston analysointi

Kun neuroverkkomallin korjausta kokeillaan testidataan, niin tällöin kuvasta 24 havaitaan, että neuroverkko tunnistaa erittäin hyvin QRS -piikit. Toisaalta saman kuvan perusteella voidaan havaita, että se reagoi huonosti ECG -käyrissä esiintyviin P- ja T -aaltoihin. Toisaalta kuvan perusteella nämä kyseiset aallot eivät datassa ole kovin selkeitä, mutta silti neuroverkko reagoi kyseisiin aaltoihin jonkin verran. Todennäköisesti suuremmalla datamäärällä ja useammalla koulutuskierrokselle pystyisimme luomaan paremman ECG -käyriä korjaavan neuroverkon.

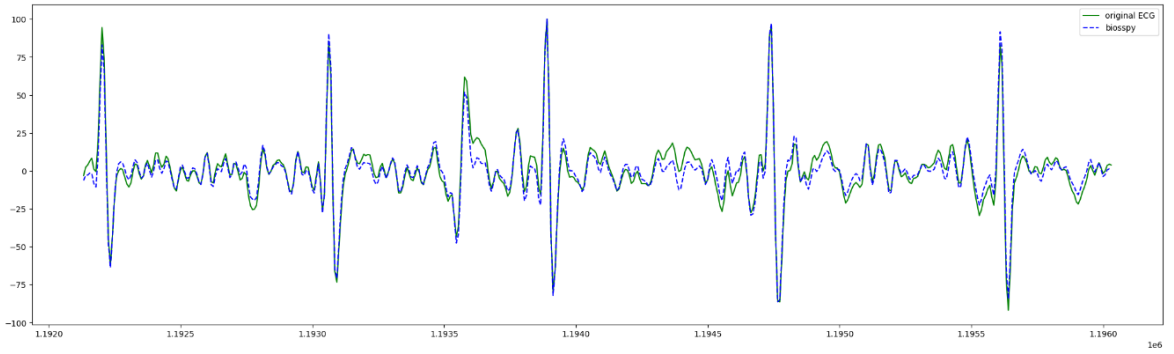


Kuva 24. Neuroverkkomallin korjaus testi datasta.

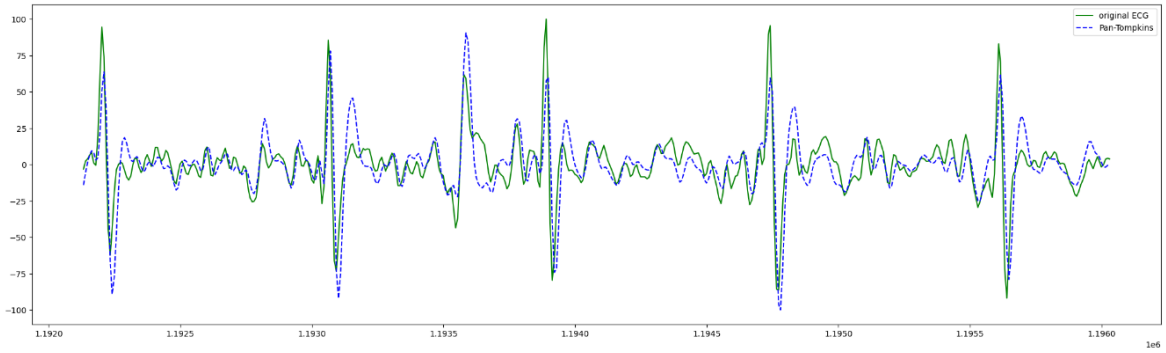
Ennen neuroverkon arviointia tutustuin sellaisiin Pythonin kirjastoihin kuin Neurokit2, Biosppy ja HeartPy. Neurokit2 sisältää erilaisia algoritmeja, joilla voidaan tarkastella ja prosessoida erilaisia lääketieteellisissä mittauksissa kerättyjä datoja. Neurokit2 sisältää sellaisen metodin kuin ecg_clean, jonka avulla voidaan käyttää erilaisia menetelmiä ECG -datan puhdistamiseksi. Kokeilin tämän avulla käyttää sellaisia puhdistusalgoritmeja, kuin Neurokit, Biosppy ja pantompkis1985. Alla on esitetty kuvat, miten nämä kyseiset algoritmit korjaavat ECG:n artefakteja erikseen kerätystä testattavasta datasta, mitä ei ole käytetty neuroverkon kouluttamisessa.



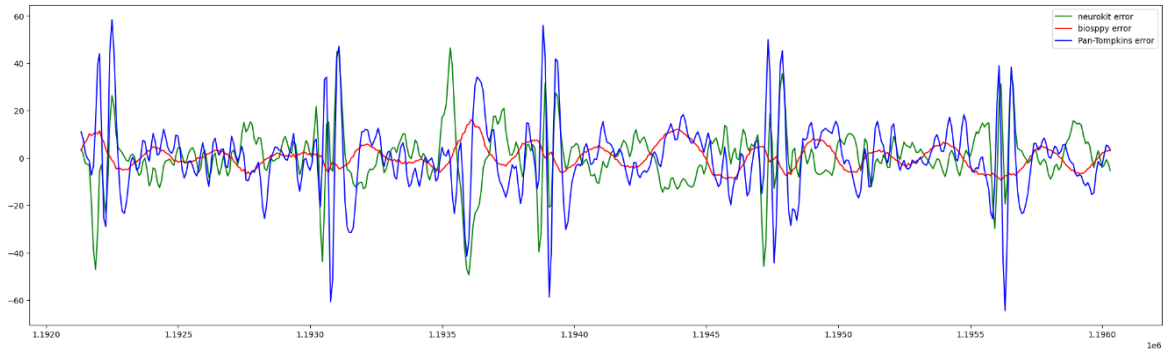
Kuva 25. Neurokitin luoma korjaus testattavasta datasta.



Kuva 26. Biosppy korjausalgoritmin luoma korjaus testattavasta datasta.

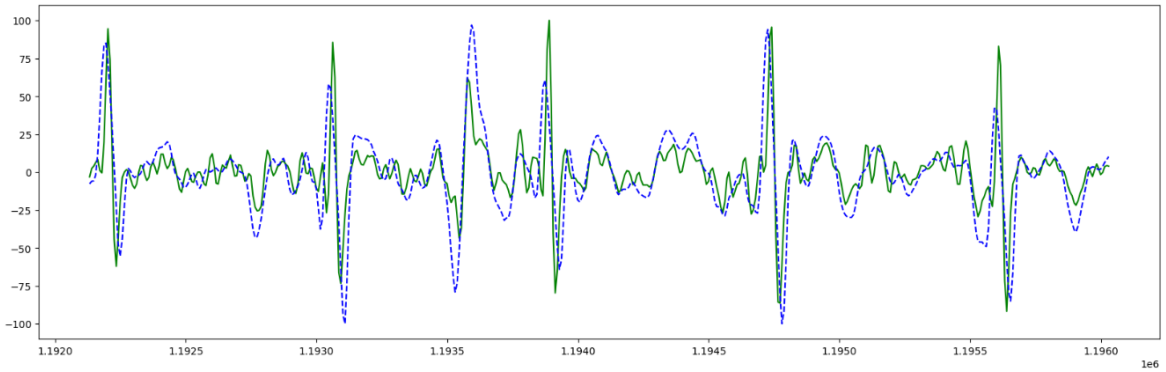


Kuva 27. Pan-Tompkins korjausalgoritmin luoma korjaus testattavasta datasta.



Kuva 28. Neurokit, Biosppy ja Pan-Tompkins korjausalgoritmien virhe testattavasta datasta.

Kuvista voidaan päätellä, että nämä kyseiset kirjaston algoritmit mukautuvat liiankin hyvin artefaktien muotoihin, eivätkä näin ollen muistuta kuvassa 2, sivulla 10 esitettyä ECG -käyrää. HeartPy -kirjaston signaalinkorjausalgoritmi pystyy korjaamaan paremmin ECG:n artefakteja, mutta sekin vaikuttaa olevan jonkin verran altis artefaktien piikeille. Ilmeisesti nämä käytössä olevat algoritmit on suunniteltu suodattamaan ECG -käyriä, jotka on kerätty lepotilassa olleilta potilailta.

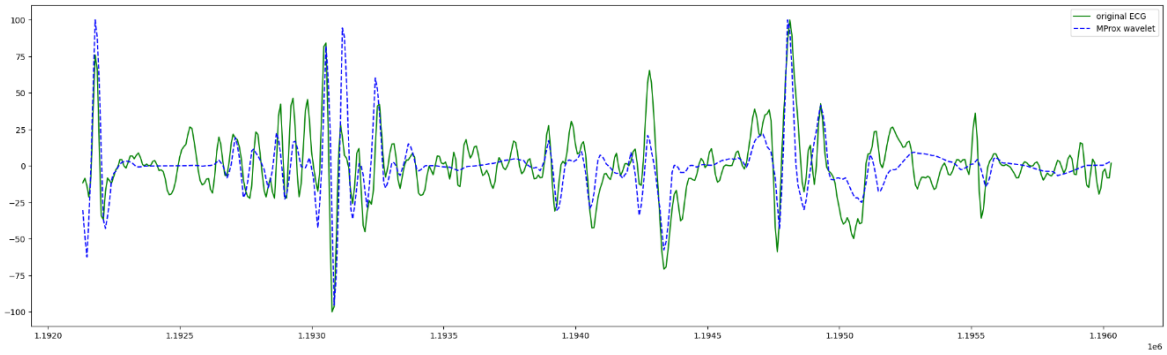


Kuva 29. HeartPy kirjaston smooth_signal algoritmin korjaus testattavasta datasta.

Jun, Nguyen, Kang, Kim, Kim ja Kim (2018) esittivät mielenkiintoisen neuroverkkoihin pohjautuvan algoritmin, joka luokitteli potilaan ECG -käyrät. Heidän algoritminsa perustui siihen, että Biosppyn signaalin käsittely algoritmi kykenee poimimaan datasta R -piikit ja

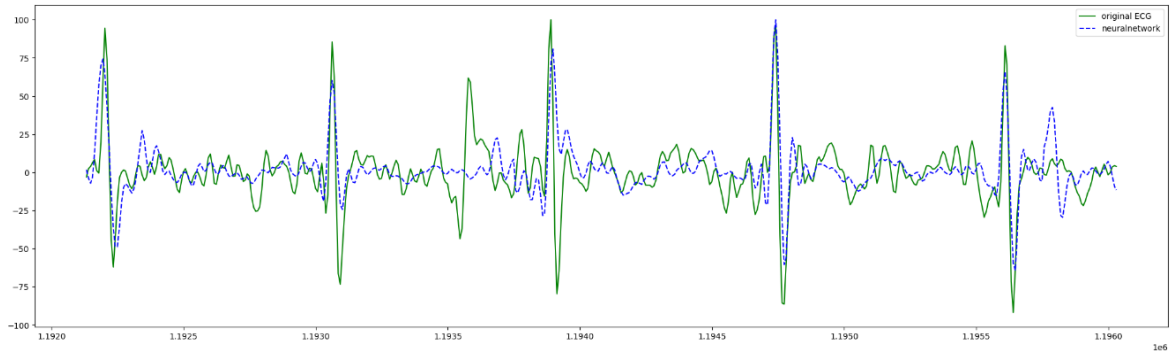
muodostavat mustavalkokuvan tämän piikin ympäristöstä, minkä jälkeen neuroverkko luokittelee kyseisen ECG -käyrän ennalta sovittuun luokkaan. Kun kokeilin kyseistä järjestelmää, havaitsin ettei kyseinen ohjelma kykene käsittelemään artefakteja sisältävää dataa kovinkaan hyvin, vaan syötettävä data on ensin normalisoitava, minkä jälkeen kyseinen Biosppyn signaalin käsittely algoritmi kykenee löytämään datasta R -piikit.

Useimmat löytämäni algoritmit joko analysoivat tai luokittelevat ECG -käyriä, tai ovat hyvin tehottomia korjaamaan ECG -käyriin liittyviä artefakteja. Eräs mahdollinen ja käytössä oleva tekniikka on käyttää wavelet signaalin käsittelyä. Wavelet -algoritmit eivät suoraan puhdista ECG -käyriä, mutta löysin MProx (2019) -nimimerkkiä käyttävän henkilön luoman ohjelman, joka on suunniteltu korjaamaan artefakteja. Kyseinen ohjelma poimii ECG -käyriin liittyvät wavelet parametrit, minkä jälkeen ohjelma poistaa jokaiselta wavelet tasolta tietyn taajuuden ylittävät arvot, minkä jälkeen wavelet -malli uudelleen rakennetaan alkuperäistä dataa vastaavaan muotoon. Vaikka tämä algoritmi korjaakin artefakteja suhteellisen hyvin, niin voimakkaat artefakti piikit vaikuttavat sen tarkkuuteen.



Kuva 30. MProx wavelettiin perustuva artefaktien korjaus testattavasta datasta.

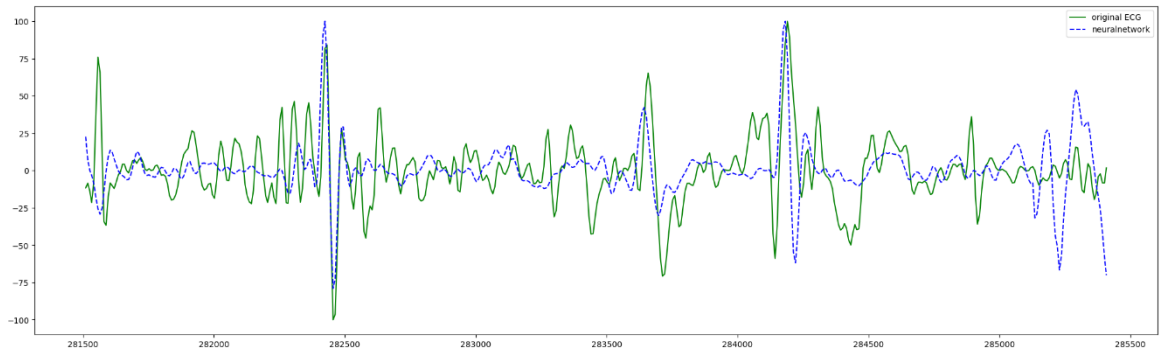
Luomallani neuroverkolla kyetään saamaan paremmin esille ECG -käyriin liittyvät piirteet, kun sensorin kosketuspinta on hyvä ja onnistutaan keräämään hyvälaatuista dataa.



Kuva 31. Neuroverkon algoritmin korjaus testattavasta datasta.

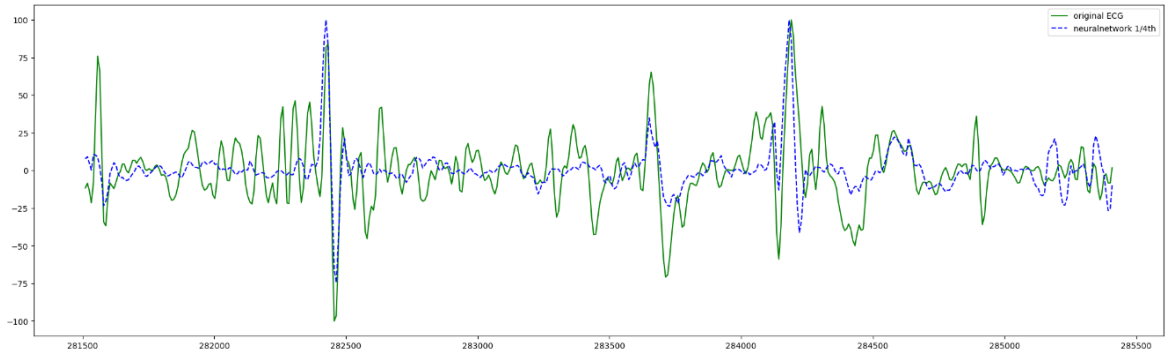
Koska neuroverkko kykeni luomaan näinkin hyvän mallin tavallisesta ECG:stä, päätin kokeilla tämän jälkeen, miten hyvin se kykenee korjaamaan huonolaatuista dataa. Sekin on

jonkin verran virhealtis voimakkaille artefaktipiikeille, minkä lisäksi neuroverkko osaa tulkita huonosti datan äärilaidoilla olevia arvoja. Tästä voidaan päätellä, että kun neuroverkkoja koulutetaan, sinne saattaa jäädä joitakin alueita, joita neuroverkko osaa tulkita huonosti.



Kuva 32. Neuroverkon artefaktien korjaus huonolaatuisesta datasta

Koska neuroverkko ei osaa korjata artefakteja koko alueelta, vaan ainoastaan tietyn datamäärän verran, jolloin jatko kehittelin tätä ajatusta siten, että testattava data ajettaisiin neuroverkon läpi joka neljäsosasekunti. Tällöin saadaan kerättyä jokaista datapistettä kohden neljä erillistä arvoa, lukuun ottamatta datan keräyshetken ensimmäisiä ja viimeisiä sekunteja. Kun näistä kerätyistä datapisteistä lasketaan mediaani ja normalisoidaan data, niin tällöin saadaan muodostettua tasaisempi ECG -käyrä.



Kuva 33. Paranneltu neutraaliverkon tulos huonolaatuisesta datasta.

Koska kunnon mallia Pythonille on erittäin hankalaa löytää, voidaan mallin paremmuutta tarkastella laskemalla testattavan datan ja korjauksen välisen erotuksen keskiarvo ja keskihajonta testattavasta datasta. Kuvassa 2 sivulla 10 esitetty ECG -käyrä on hyvin tasainen, minkä vuoksi olemme kiinnostuneita tietämään, kuinka tasaisesti algoritmit kykenevät korjaamaan ECG -käyrän. Kun määritelemme tasaisuuden yhtälö kaavalla

$$T([x_1, \cdots, x_n]) = \frac{1}{\max\left\{1, \left|\max_{j=1..n}\{x_j\} - \min_{j=1..n}\{x_j\}\right|\right\}} \cdot \sum_{i=1}^{n-1} |x_{i+1} - x_i|$$

missä x_i:t ovat ECG -käyrän arvoja ja n on testattavan datan havaintojen lukumäärä. Tämä algoritmi laskee peräkkäisten datapisteiden väliset etäisyydet yhteen ja jakaa ne

suurimman ja alimman datapisteiden välisen etäisyyden. Esimerkiksi jos kaikki datapisteet saisivat arvon c, niin tällöin tasaisuudeksi saataisiin arvo nolla. Mikäli taas datapisteet muodostavat (laskevan/nousevan) suoran, toisen asteen peruspolynomin tai kolmannen asteen peruspolynomin, niin tasaisuuden arvo asettuu lähelle yhtä. Mikäli datassa esiintyy paljon piikkejä, niin sitä isompia arvoja tämä tasaisuuden yhtälö antaa. Siten voidaan muodostaa alla oleva taulukko eri ECG -algoritmeille ja vertailla niiden tuottamia tuloksia.

Malli	**Virheen keskiarvo**	**Virheen hajonta**	**Tasaisuus**
Neuroverkko	-2,46	18,56	2926,21
Neurokit	0,44	11,35	7616,82
Biosppy	-0,1	6,29	9987,26
Pan-Tompkins	0,38	16,54	8936,89
MProx	0,25	19,91	3358,23

Näiden lukujen perusteella Biosppy ei vaikuta kovinkaan varteenotettavalta vaihtoehdolta. Vaikka sen virheiden keskiarvo ja hajonta on pientä, niin tasaisuuden mittari osoittaa, että kyseinen algoritmi seuraa liiankin tiukasti kaikkia ECG -käyrässä esiintyviä artefakteja. MProx -algoritmi vaikuttaa mielenkiintoiselta algoritmilta, koska tasaisuuden mittarin mukaan se on suhteellisen tasainen algoritmi. Toisaalta kuvasta 30 on kuitenkin nähtävissä, että se reagoi herkästi voimakkaisiin artefakteihin. Neurokit antaa tasaisuuden mittarin mukaan suhteellisen tasaisia tuloksia ja hajontaluku osoittaisi, ettei kyseinen algoritmi seuraa liiankin tiukasti ECG -käyrän artefakteja. Kuvasta 25 on havaittavissa, että se antaa parempia estimaatteja ECG:n rakenteesta kuin MProx. Näiden mittausten ja arviointien perusteella luotu neuroverkkomalli ja Neurokit antavat parhaimmat arviot ECG -käyrän muodosta, vaikkakin Neurokit on turhankin altis reagoimaan joihinkin artefakteihin.

6 Tulosten tarkastelu

Tämän opinnäytetyön tarkoituksena oli selvittää, miten usealla sensorilla voidaan kerätä ECG dataa ja miten neuroverkko voidaan kouluttaa korjaamaan ECG -käyrässä esiintyviä artefakteja. Työssäni suunnittelin tarkoituksella neuroverkon niin pieneksi, että sen voisi tarpeen mukaan ohjelmoida toimimaan mobiililaitteessa. ECG -käyristä ja niihin liittyvistä artefakteista löytyi paljon tutkimustietoa, minkä lisäksi on jonkin verran julkaistu tutkimuksia, joissa on käsitelty artefaktien poistamista datasta algoritmisin keinoin. ECG -käyrien tutkimuksissa on käytetty jonkin verran neuroverkkoja, mutta näiden tutkimusten pääasiallisena tavoitteena on ollut kouluttaa neuroverkot tunnistamaan sydänsairauksia tai havainnoimaan R -piikkejä, eikä niinkään puhdistamaan artefakteja ECG -käyristä.

Myös kiristyvä mobiiliturva aiheuttaa jonkin verran haasteita, kun ohjelmoidaan matkapuhelimien ja sensoreiden välistä viestintää. Syksyllä 2021 Android kiristi laitteidensa turvallisuutta, minkä vuoksi Movesensen tarjoamaa ohjelmakoodia ei kyetty ajamaan kunnolla päivitetyssä matkapuhelimessa. Jonkin verran datan keruuta hidasti myös se, ettei Movesensen sensoreita voinut päivittää suoraan tai jos se oli mahdollista, niin en ainakaan löytänyt ohjeita siihen, miten tämä on mahdollista tehdä. Tämän vuoksi kehitin toisen tavan synkronoida useiden sensoreiden data, käyttämällä hyväksi tallennusjärjestelmän aikaleimausta. Näistä ongelmista huolimatta onnistuin kuitenkin keräämään sen verran dataa, että neuroverkon kouluttaminen oli mahdollista ja kykenin vertailemaan erilaisia suodattimia.

Neuroverkon kouluttamisessa oli jonkin verran haasteita, kun kokeilin erilaisia neuroverkkoja ja niiden kouluttamista. Yleensä ECG -käyriä mitataan, kun henkilö on lepotilassa, mutta löysin sattumalta Physionetistä aineiston (PhysioNet, 2015; Goldberg & all. 2000) jossa koehenkilön dataa kerätään useiden sensoreiden avulla, kun hän suorittaa kuormittavia tehtäviä. Kun vertailin omaa aineistoa ja sen perusteella koulutettua neuroverkkoa, havaitsin että koulutettu neuroverkko ei kyennyt suodattamaan artefakteja. Kun poimin satunnaisesti eri laitteilla kerättyä ECG -dataa ja kiihtyvyysdataa, onnistuin kehittämään neuroverkon, joka kykeni korjaamaan yllättävän hyvin artefakteja vertailuaineistosta. Ilmeisesti sensoreiden keräystapa vaikuttaa jonkin verran aineiston laatuun, minkä vuoksi tarvitaan ECG -dataa useista sensorilähteistä, jotta sillä voidaan kouluttaa neuroverkko tunnistamaan artefakteja.

Toinen neuroverkon kouluttamiseen ja dataan liittyvä haaste koskettaa datan jakamista koulutus-, validointi- ja testidataan. Alkuperäinen ajatus oli poimia R -piikkejä datasta ja pyrkiä kouluttamaan neuroverkko siistimään ECG -käyrä tämän piikin ympäristöstä. Tutki-

musta tehdessäni minulle juolahti ajatus, että miksi meidän tulisi tehdä näin, koska periaatteessa saatamme joutua käsittelemään sotkuista dataa ja R -piikit saattavat sijoittua hieman eri paikkoihin. Tämän takia päätin, että poimin aineistosta ensiksi validointi- ja testidatan ja lukitsen nämä kohdat datasta. Tämän jälkeen valitsin satunnaisesti kymmeniä tuhansia sekunnin pituisia pätkiä koulutustarkoitusta varten. Ajatuksenani oli, että ei ole mitään väliä, vaikka valitsisimme lähes saman kohdan koulutusta varten, koska hyvin suurella todennäköisyydellä aloituskohta ja lopetuskohta poikkeavat alkuperäisestä poimitusta datasta.

Kolmas neuroverkkoihin liittyvä haaste koskee koulutusvaihetta, mikä neuroverkoilla on yleensä hidas prosessi. Varsinkin kun testataan ja kokeillaan erilaisia neuroverkkovaihtoehtoja. Yllätykseksenі havaitsin, että liian suuri neuroverkko onkin huonompi kuin pienempi neuroverkko. Lisäksi neuroverkon kouluttaminen ECG -datalla vaatii erittäin paljon muistia. Erityisesti silloin kun ECG -data on normalisoitava ennen kouluttamista. Esimerkiksi tavallinen toimistokone ei kyennyt prosessoimaan koulutusdataa ja kouluttamaan neuroverkkoja muistiongelmien vuoksi. Tämä osoittautui melko isoksi haasteeksi, kun jouduin korjauttamaan koneeni emolevyn.

Alkuperäisenä ajatuksenani oli kehittää R -piikkien, suodattimien ja neuroverkon avulla artefakteja korjaava algoritmi. Lopulta tutkimuksen ja kokeilujen perusteella päädyin yksinkertaisempaan neuroverkkoratkaisuun, joka antoi lopulta yllättävän hyviä lopputuloksia. Toisaalta mikäli data on sotkuista, niin kehitetyllä neuroverkolla oli vaikeuksia korjata tämän kaltaista dataa. Toisaalta keräämäni tulokset vaikuttavat siltä, että mikäli keräisin dataa useammalta henkilöltä ja eri sensoreilta, niin tällöin kykenisin muodostamaan neuroverkon, joka kykenisi paremmin korjaamaan ECG -käyrissä esiintyviä artefakteja. Tätä kyseistä neuroverkkomallia voisi myös jatkokehittää ja testata erilaisilla neuroverkkokerroksilla. Tämän lisäksi kouluttamaan neuroverkon laadunmittaustapaa voisi kehittää paremmaksi, koska neuroverkkoa kouluttaessa ilmeni, että neliösumma ja neuroverkkojen tarkkuusmittari eivät ole parhaimpia mittareita arvioimaan neuroverkon tarkkuutta, vaikkakin niiden avulla kyettiin kouluttamaan neuroverkko.

Lähteet

Aalto yliopisto, 2019. Python-oppimateriaali (CHEM-A2600). Luettavissa: https://mycourses.aalto.fi/mod/book/view.php?id=442395&chapterid=2510&lang=fi. Luettu: 16.4.2023.

Abdulla, R., Bonney, W., Khalid, O. & Awad, S. 2016. Pediatric Electrocardiography: An Algorithm Approach to Interpretation. Springer International Publishing Switzerland. Luettavissa: https://link.springer.com/content/pdf/10.1007/978-3-319-26258-1.pdf. Luettu: 4.7.2021.

Ahmed, S., Hilal-Alnaqbi, A., Hemairy, M.A. & Ahmad, M.A. 2018. ECG Abnormality Detection Algortihm. International Journal of Advanced Computer Science and Applications, Vol. 9. No. 8. Luettavissa: https://pdfs.semanticscholar.org/39d0/47497893d5d784a649aa00a9069bd0e0b2c9.pdf. Luettu: 28.6.2021.

Atlas of Human Cardiac Anatomy. 2021. Your Heart and the Cardiovascular System. University of Minnesota. Luettavissa: http://www.vhlab.umn.edu/atlas/physiology-tutorial/index.shtml. Luettu: 28.7.2021.

Brownlee, J. 2019. A Gentle Introduction to the Rectified Linear Unit (ReLU). Luettavissa: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. Luettu: 23.4.2023.

Chanwimalueang, T., von Rosenberg, W. & Mandic, D.P. 2015. Enabling R-peak Detection in Wearable ECG: Combining Matched Filtering and Hilbert Transform. IEEE International Conference on Digital Signal Processing. Luettavissa: https://ieeexplore.ieee.org/abstract/document/7251845. Luettu: 20.7.2021.

Chollet, F. & all. 2015. Keras. Luettavissa: https://github.com/keras-team/keras. Luettu: 16.2.2023.

Choubey, V. 2023. Activation functions in neural network: steps and implementation. Luettavissa: https://medium.com/codex/activation-functions-in-neural-network-steps-and-implementation-df2e4c858c21. Luettu: 18.2.2023.

Debbabi, N., Asmi, S.E. & Arfa, H. 2010. Correction of ECG baseline wander Application to the Pan & Tompkins QRS detection algorithm. Luettavissa: https://ieeexplore.ieee.org/abstract/document/5654714. Luettu: 4.7.2021.

Espoon matikkamaa, 2020. Matemaattisia vakioita, jotka jokaisen tulisi tuntea. Luettavissa: https://www.espoonmatikkamaa.fi/2020/03/26/matemaattisia-vakioita-jotka-jokaisen-tulisi-tuntea/. Luettu: 28.6.2023.

Ghaleb, F.A., Kamat, M.B., Salleh, M., Rohani, M.F. & Razak, S.A. 2018. Two-stage motion artefact reduction algorithm for electrocardiogram using weighted adaptive noise cancelling and recursive Hampel filter. Luettavissa: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207176. Luettu: 20.7.2021.

Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P.C., Mark, R., Mietus, J.E., Moody, G.B., Peng, C.K. and Stanley, H.E., 2000. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

Goy, J-J., Staufer, J-C. & Schlaepfer. 2013. Electrocardiography (ECG). Bentham Science Publisher. Luettavissa: https://ebookcentral.proquest.com/lib/helsinki-ebooks/detail.action?pq-origsite=primo&docID=1310813#goto_toc. Luettu: 26.7.2021.

He, T., Clifford, G., & Tarassenko, L. 2006. Application of independent component analysis in removing artefacts from the electrocardiogram. Neural Computing & Applications, 15(2), sivut 105-116. Luettavissa: https://link.springer.com/article/10.1007/s00521-005-0013-y. Luettu: 28.6.2021.

Helsingin yliopisto. 2023. Käyttöjärjestelmä ja käyttöliittymä. Luettavissa: https://blogs.helsinki.fi/opiskelijan-digitaidot/1-tietokoneen-kayton-perusteet/1-1-tietokoneen-toimintaperiaate/kayttojarjestelma-ja-kayttoliittyma/. Luettu: 16.4.2023.

Jenkins, D. 2009. A (not so) brief history of electrocardiography. ECG library. Luettavissa: https://ecglibrary.com/ecghist.html. Luettu: 25.7.2021.

Jun, T.J., Nguyen, H.M., Kang, D., Kim, D.,Kim, D. & Kim, Y. 2018. ECG arrhythmia classification using a 2-D convolutional neural network. Luettavissa: https://arxiv.org/abs/1804.06812. Luettu: 5.4.2023.

Kathuria, A. 2023. Intro to optimization in deep learning: Gradient Descent. Luettavissa: https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/. Luettu: 1.7.2023.

Keim, R. 2020. Understanding Local Minima in Neural-Network Training. Luettavissa: https://www.allaboutcircuits.com/technical-articles/understanding-local-minima-in-neural-network-training/. Luettu: 29.6.2023.

Kelleher, J.D. 2020. Syväoppiminen. Suomentanut Pietiläinen, K. Libris/ Painoliber Oy, Helsinki.

Keras. 2023a. Droput layer. Luettavissa: https://keras.io/api/layers/regularization_layers/dropout/. Luettu: 19.2.2023.

Keras. 2023b. Flatten layer. Luettavissa: https://keras.io/api/layers/reshaping_layers/flatten/. Luettu: 19.2.2023.

Keras. 2023c. Optimizers. Luettavissa: https://keras.io/api/optimizers/. Luettu: 19.2.2023.

Keras. 2023d. Losses. Luettavissa: https://keras.io/api/losses/. Luettu: 19.2.2023.

Keras. 2023e. Layer Activation functions. Luettavissa: https://keras.io/api/layers/activations/. Luettu: 29.6.2023.

Kotimikro, 2019. Näin löydät laitteesi MAC-osoitteen. Luettavissa: https://kotimikro.fi/internet/verkko/nain-loydat-laitteesi-mac-osoitteen. Luettu: 23.4.2023.

Köhler,B-U., Henning, C & Orglmeister, R. 2002. The Principles of Software QRS Detection: Reviewing and Comparing Algorithms for Detecting this Important ECG Wafeworm. IEEE Engineering in Medicine and Biology. January. s. 42-57. Luettavissa: https://ieeexplore.ieee.org/abstract/document/993193. Luettu: 20.7.2021.

Maheshkar, S. 2022. How to compare Keras pptimizers in Tensorflow for deep learning. Luettavissa: https://wandb.ai/sauravm/Optimizers/reports/How-to-Compare-Keras-Optimizers-in-Tensorflow-for-Deep-Learning--VmlldzoxNjU1OTA4. Luettu: 21.2.2023.

Maksutov, R. 2018. Deep study of a not very deep neural network. Part 3b: Choosing an optimizer. Luettavissa: https://rinat-maksutov.medium.com/deep-study-of-a-not-very-deep-neural-network-part-3b-choosing-an-optimizer-de8965aaf1ff. Luettu: 20.2.2023.

Marini, F., Bucci, R., Magri, A.L. & Magri, A.D. 2008. Artificial neural networks in chemometrics: History, examples and perspectives. Microchemical Journal, Vol. 88, Issue 2, s.

178-185. Luettavissa: https://www.sciencedirect.com/science/article/abs/pii/S0026265X07001403. Luettu: 3.10.2021.

Math is fun, 2023. E (Euler's number). Luettavissa: https://www.mathsisfun.com/numbers/e-eulers-number.html. Luettu; 23.4.2023.

Matthews, B. E. Circulatory System, anatomy. Encyclopadia Britannica. Luettavissa: https://www.britannica.com/science/circulatory-system. Luettu: 28.7.2021.

Mazomenos, E. B., Biswas, D., Acharyya, A., Chen, T., Maharatna, K. Rosengarten, J., Morgan, J. & Curzen, N. 2013. A Low-Complexity ECG Feature Extraction Algorithm for Mobile Healthcare Applications. IEEE Journal of Biomedical and Health Informatics, Vol. 17, No. 2, sivut 459 – 469. Luettavissa: https://ieeexplore.ieee.org/abstract/document/6420852. Luettu: 30.7.2021.

Mediasignal. 2023. Ohjelmistointegraatio tehostaa ja yhtenäistää liiketoimintaa. Luettavissa: https://mediasignal.fi/blogi/ohjelmistointegraation-suunnittelu-rest-api-soap-xml-json-what/. Luettu: 16.4.2023.

Movesense. 2021. Movesense System Overview. Luettavissa: http://www.movesense.com/docs/system/system_overview/. Luettu: 27.6.2021.

MProx. 2019. Wavelet-denoising. Luettavissa: https://github.com/MProx/Wavelet-denoising. Luettu 5.4.2023.

Nair, M.A. 2010. ECG Feature Extraction using Time Frequency
Analysis. In: Sobh T., Elleithy K. (eds) Innovations in Computing Sciences and Software Engineering, s.461-466. Springer, Dordrecht. Luettavissa: https://link.springer.com/chapter/10.1007/978-90-481-9112-3_78. Luettu: 4.7.2021.

NBCI. 2019. How does the blood circulatory system work? Luettavissa: https://www.ncbi.nlm.nih.gov/books/NBK279250/. Luettu: 28.7.2021.

NobelPrize.org. 2021. The Nobel Prize in Physiology or Medicine 1924. The Nobel Prize. Luettavissa: https://www.nobelprize.org/prizes/medicine/1924/summary/. Luettu: 27.7.2021.

Nykänen, O. 2021. Weka-aloitusohje koneoppimisen kokeiluiden tekemiseen ilman ohjelmointia. Luettavissa: https://blogs.tuni.fi/ai-lahettilas/tekninen-aloitusohje/weka-aloitusohje-koneoppimisen-kokeiluiden-tekemiseen-ilman-ohjelmointia/. Luettu: 1.7.2023.

Pan, J. & Tompkins, W.J. 1985. A real-time QRS Detection Algorithm. IEEETransactions on Biomedical Engineering, Vol. BME-32, No. 3. s. 230-236. Luettavissa: A Real-Time QRS Detection Algorithm | IEEE Journals & Magazine | IEEE Xplore. Luettu: 30.9.2021.

Peng, Z., Wang, G., Jiang, H. & Meng, S. 2017. Research and improvement of ECG compression algorithm based on EZW. Computer Methods and Programs in Biomedicine 145, s. 157-166. Luettavissa: https://www.sciencedirect.com/science/article/pii/S0169260716310756. Luettu: 4.7.2021.

PhysioNet. 2015. Motion Artifact Contaminated ECG Database. Luettavissa: https://physionet.org/content/macecgdb/1.0.0/. Luettu: 7.4.2023.

Poutanen, T. & Hiippala, A. 2016. Miten tulkitsen lapsen EKG:n? Suomen lääkärilehti, No. 45, VSK 71. s. 2875-2881a. Luettavissa: https://helda.helsinki.fi//bitstream/handle/10138/229991/SLL452016_2875.pdf?sequence=1. Luettu: 27.7.2021.

PyTorch. 2023. Leakyrelu. Luettavissa: https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html. Luettu: 23.4.2023.

Rizzoli, A. 2022. An Introductory Guide to Quality Training Data for Machine Learning. Luettavissa: https://www.v7labs.com/blog/quality-training-data-for-machine-learning-guide. Luettu: 1.7.2023.

Salam, A. M. 2019. The Invention of Electrocardiography Machine. Heart Views, Vol. 20, No. 4. s. 181-183. Luettavissa: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6881865/. Luettu: 25.7.2021.

Sarang, P. 2020. Artificial Neural Networks with TensorFlow 2: ANN Architecture Machine Learning Projects. Apress. Luettavissa: https://learning.oreilly.com/library/view/artificial-neural-networks/9781484261507/. Luettu: 4.10.2021.

Sivaraks, H. & Ratanamahatana, C. A. 2015. Robust and Accurate Anomaly Detection in ECG Artifacts Using Time Series Motif Discovery. Computational and Mathematical Methods in Medicine. Luettavissa: https://www.hindawi.com/journals/cmmm/2015/453214/. Luettu: 23.9.2021.

SQLite. 2023. About SQLite. Luettavissa: https://sqlite.org/about.html. Luettu: 23.4.2023.

Story of mathematics. 2003. Amplitude | Definition & Meaning. Luettavissa: https://www.storyofmathematics.com/glossary/amplitude/. Luettu: 7.3.2023.

Stroobandt, R. X, Barold, S. S. & Sinnaeve, A. F. 2016. ECG from basics to essentials: step by step. Luettavissa: https://ebookcentral-proquest-com.libproxy.helsinki.fi/lib/helsinki-ebooks/reader.action?docID=4093349. Luettu: 27.7.2021.

Study mind. 2023. Depolarisation and Repolarisation in the Action Potential (A-level Biology). Luettavissa: https://studymind.co.uk/notes/depolarisation-and-repolarisation-in-the-action-potential/. Luettu: 17.4.2023.

Sugomori, Y., Kaluža, B., Soares, F. M., Souza, A. M. F., Kaluza, B. & Soares, F. M. 2017. Deep learning: Practical Neural Networks with Java. Packt Publishing Ltd. Luettavissa: https://learning.oreilly.com/library/view/deep-learning-practical/9781788470315/?ar=. Luettu: 3.10.2021.

Suunto. 2021. Movesense iOS and Android Libraries. Luettavissa: https://bitbucket.org/suunto/movesense-mobile-lib/src/master/. Luettu: 27.6.2021.

Suunto. 2023a. Movesense system overview. Luettavissa: https://www.movesense.com/docs/system/system_overview/. Luettu: 28.6.2023.

Suunto. 2023b. Mobile software development overview. Luettavissa: https://www.movesense.com/docs/mobile/mobile_sw_overview/. Luettu: 28.6.2023.

Syvänne, M. & Hekkala, A-M. 2019. Sydän- ja verisuonitautien tutkimukset. Sydän.fi. Luettavissa: https://sydan.fi/fakta/sydan-ja-verisuonitautien-tutkimukset/. Luettu: 27.7.2021.

Tiusanen, M.J. & Pastell, M. 2016. Simple online algorithm for Detecting Cow's ECG beat-to-beat interval using a microcontroller. International Commission of Agricultural and Biosystems Engineering. E-Journal - CIGRI, Vol. 18, No. 1, s. 411-418 Luettavissa: https://researchportal.helsinki.fi/en/publications/simple-online-algorithm-for-detecting-cows-ecg-beat-to-beat-inter. Luettu: 4.7.2021.

Tobón-Cardona, M., Kenttä, T. Porthan, K., Tikkanen, J.T., Oikarinen, L., Viitasalo, M., Salomaa, V., Huikuri, H.V., Junttila, J.M. & Seppänen, T. 2018. Waveform prototype-

based feature learning for automatic detection of the early repolarization pattern in ECG signals. Physiological Measurement, Vol. 39, No. 11. Luettavissa: https://iop-science.iop.org/article/10.1088/1361-6579/aaecef/meta. Luettu: 4.7.2021.

Townsend, K. 2014. GATT. Luettavissa: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. Luettu: 16.4.2023.

ValueFrame. 2023. REST-rajapinnan käyttö. Luettavissa: https://support.valueframe.fi/support/solutions/articles/77000527771-rest-rajapinnan-k%C3%A4ytt%C3%B6. Luettu: 16.4.2023.

Vollmer, M., Bläsing, D., Reiser, J. E., Nisser, M. & Buder, A. 2022. Simultaneous physiological measurements with five devices at different cognitive and physical loads (version 1.0.1). PhysioNet. Luettavissa: https://doi.org/10.13026/zhns-t386. Luettu: 15.9.2022.

Warr, K. 2019. Strengthening Deep Neural Networks. O'Reilly Media Inc. Luettavissa: https://learning.oreilly.com/library/view/strengthening-deep-neural/9781492044949/. Luettu: 4.10.2021.

Wikimedia. 2021. File:Ecg.png. Luettavissa https://commons.wikimedia.org/wiki/File:Ecg.png. Luettu: 31.5.2021.

Wolfram Mathworld, 2023a. Maximum. Luettavissa: https://mathworld.wolfram.com/Maximum.html. Luettu: 7.7.2023.

Wolfram Mathworld, 2023b. Median. Luettavissa: https://mathworld.wolfram.com/Median.html. Luettu: 7.7.2023.

Wolfram Mathworld, 2023c. Minimum. Luettavissa: https://mathworld.wolfram.com/Minimum.html. Luettu: 7.7.2023.

Wolfram Mathworld, 2023d. Weighted mean. Luettavissa: https://mathworld.wolfram.com/WeightedMean.html. Luettu: 7.7.2023.

Wong, S. 2019. How Willem Einthoven gave doctors a window on the heart. New Scientist. Luettavissa: https://www.newscientist.com/article/2203921-how-willem-einthoven-gave-doctors-a-window-on-the-heart/. Luettu: 25.7.2021.

Ye, A. 2020. A guide to neural network layers with applications in Keras. Luettavissa: https://medium.com/analytics-vidhya/a-guide-to-neural-network-layers-with-applications-in-keraköhs-40ccb7ebb57a. Luettu: 21.2.2023.

Liitteet

Liite 1. Datan normalisointi

```
def scale_data(d, V=256):
    data = d.copy()
    data = d.copy()
    n = len(data)
    if (n<V):
        return data / np.max(abs(data))
    width = int(V/2)
    m1 = np.ones(width)*np.max(abs(data[0:width]))
    m2 = np.ones(width)*np.max(abs(data[n-width:n]))
    m = np.zeros((2,n-V))
    h = abs(data[0:n-V])
    for i in range(1,V):
        m[0,:] = h
        m[1,:] = abs(data[i:n-V+i])
        h = np.max(m,axis=0)
    m = np.max(m,  axis=0)
    y = np.append(m1,m)
    y = np.append(y,m2)
    y = data / y *100
    return y
def calculate_median(ecg):
    V= 60
    data = ecg.copy()
    n = len(data)
    if (n<V):
        return np.median(data)*np.ones(len(data))
    width = int(V/2)
    m1 = np.ones(width)*np.median(data[0:width])
    m2 = np.ones(width)*np.median(data[n-width:n])
    m = np.zeros((V,n-V))
    m[0,:] = data[0:n-V]
    for i in range(0,V):
        m[i,:] = data[i:n-V+i]
    m=np.median(m,  axis=0)
    y = np.append(m1,m)
    y = np.append(y,m2)
    return y
def medianf(ecg):
    data = ecg.copy()
    m = calculate_median(ecg)
    return data - m

def normalization(ecg):
    return scale_data(medianf(ecg))
```

Liite 2. Datan synkronointi

```
# Check intervals and combine them
i=0
j=0
add_len = 0

time_beg = list()
time_end = list()

while i<len(t_beg) and j<len(t_end):

    index = (t_beg>=(t_beg[i]-df)) * (t_beg <= (t_beg[i] + df))
    t_b = t_beg[index==True]
    index = (t_end>=(t_end[i]-df)) * (t_end <= (t_end[i] + df))
    t_e = t_end[index==True]


    if ( len(t_b)>0 and len(t_e)>0):
        a = np.max(t_b)
        b = np.min(t_e)


        if add_len>0 and b < time_beg[add_len-1]:
            j = j + len(t_e)
        elif add_len>0 and a < time_end[add_len-1]:
            i = i + len(t_b)
        else:
            print( a, " -- ", b)

            time_beg.append(a)
            time_end.append(b)

            add_len = add_len + 1
            i = i + len(t_b)
            j = j + len(t_e)

    else:
        i = i + 1
        j=j+1



show_plot = True
A=5000
B = A + 320
df = 5000
fs=521

ecg_limit = 0.75
upper_limit = 10
lower_limit = -10
upper_limit2 = 30
lower_limit2 = -30
upper_limit3 = 20
lower_limit3 = -20
```

```
new_time1 = list()
new_time2 = list()
new_time3 = list()
new_data1 = list()
new_data2 = list()
new_data3 = list()
new_time_acc  = list()
new_data_x_acc  = list()
new_data_y_acc  = list()
new_data_z_acc  = list()

org_time1 = list()
org_time2 = list()
org_time3 = list()

data_limit = 0

for i in range(0, len(time_beg)):

    if (show_plot):
        a = org_t1[A]
        b = org_t1[B]
        ind1 =
((org_t1>=a)*(org_t1<=b)*np.arange(1,len(org_t1)+1)).astype(int)
        ind2 =
((org_t2>=a)*(org_t2<=b)*np.arange(1,len(org_t2)+1)).astype(int)
        ind3 =
((org_t3>=a)*(org_t3<=b)*np.arange(1,len(org_t3)+1)).astype(int)
        ind1 = ind1[ind1>0]-1
        ind2 = ind2[ind2>0]-1
        ind3 = ind3[ind3>0]-1

        dh1 = normalization(data1)
        dh2 = normalization(data2)
        dh3 = normalization(data3)

        plt.plot(org_t1[ind1], dh1[ind1],'r-', org_t2[ind2],
dh2[ind2],'b-', org_t3[ind3], dh3[ind3],'g-')
        plt.show()

        dh1 = None
        dh2 = None
        dh3 = None

    ind = (timestamp_ecg1>=(time_beg[i]-df)) *
(timestamp_ecg1<=(time_end[i]+df))
    org_t1 = timestamp_ecg1[ind]
    sen_t1 = t_ecg1[ind]
    data1  = ecg1[ind]

    ind = (timestamp_ecg2>=time_beg[i]-df) *
(timestamp_ecg2<=time_end[i]+df)
    org_t2 = timestamp_ecg2[ind]
    sen_t2 = t_ecg2[ind]
    data2  = ecg2[ind]

    ind = (timestamp_ecg3>=time_beg[i]-df) *
(timestamp_ecg3<=time_end[i]+df)
    org_t3 = timestamp_ecg3[ind]
```

```
    sen_t3 = t_ecg3[ind]
    data3  = ecg3[ind]

    ind = (timestamp_acc>=(time_beg[i]-df)) *
(timestamp_acc<=(time_end[i]+df))
    org_acc = timestamp_acc[ind]
    sen_acc = t_acc[ind]
    data_x_acc  = x_acc[ind]
    data_y_acc  = y_acc[ind]
    data_z_acc  = z_acc[ind]

    if (data_limit>0):
        org_t1 = org_t1[0:data_limit]
        sen_t1 = sen_t1[0:data_limit]
        data1  = data1[0:data_limit]

        org_t2 = org_t2[0:data_limit]
        sen_t2 = sen_t2[0:data_limit]
        data2  = data2[0:data_limit]

        org_t3 = org_t3[0:data_limit]
        sen_t3 = sen_t3[0:data_limit]
        data3  = data3[0:data_limit]

        m = min(len(org_acc), data_limit)
        org_acc = org_acc[0:m]
        sen_acc = sen_acc[0:m]
        data_x_acc  = data_x_acc[0:m]
        data_y_acc  = data_y_acc[0:m]
        data_z_acc  = data_z_acc[0:m]

    ind = None
    diff_1 = min( (org_t1 + sen_t1 - min(org_t1) -
min(sen_t1))[1:] )
    diff_2 = min( (org_t2 + sen_t2 - min(org_t2) - min(sen_t2)
)[1:] )
    diff_3 = min( (org_t3 + sen_t3 - min(org_t3) - min(sen_t3)
)[1:] )
    diff_acc = min( (org_acc + sen_acc - min(org_acc) -
min(sen_acc))[1:] )

    t1= (min(org_t1) -  diff_1 + (sen_t1-min(sen_t1)))
    t2= (min(org_t2) - diff_2 + (sen_t2-min(sen_t2)))
    t3= (min(org_t3) - diff_3 + (sen_t3-min(sen_t3)))

    a = t1[A]
    b = t1[B]

    t_a = min(org_acc) -  diff_acc + (sen_acc-min(sen_acc))

    new_time1.append(t1)
    new_time2.append(t2)
    new_time3.append(t3)


    if (show_plot ):
        ind1 =
((t1>=a)*(t1<=b)*np.arange(1,len(t1)+1)).astype(int)
```

```
        ind2 =
((t2>=a)*(t2<=b)*np.arange(1,len(t2)+1)).astype(int)
        ind3 =
((t3>=a)*(t3<=b)*np.arange(1,len(t3)+1)).astype(int)
        ind1 = ind1[ind1>0]-1
        ind2 = ind2[ind2>0]-1
        ind3 = ind3[ind3>0]-1

        dh1 = normalization(data1)
        dh2 = normalization(data2)
        dh3 = normalization(data3)

        plt.plot(t1[ind1], dh1[ind1],'r-', t2[ind2], dh2[ind2],'b-
', t3[ind3], dh3[ind3],'g-')
        plt.show()

        dh1 = None
        dh2 = None
        dh3 = None

    w1 = normalization(data1)
    w2 = normalization(data2)
    w3 = normalization(data3)

    _, bio1 = nk2.ecg_peaks(w1, sampling_rate=128)
    _, bio2 = nk2.ecg_peaks(w2, sampling_rate=128)
    _, bio3 = nk2.ecg_peaks(w3, sampling_rate=128)

    bio1 = bio1['ECG_R_Peaks']
    bio2 = bio2['ECG_R_Peaks']
    bio3 = bio3['ECG_R_Peaks']

    dt1 = t1[bio1]
    dt2 = t2[bio2]
    dt3 = t3[bio3]

    # align timelines
    for j in range(0,1):
        ind2 = find_nearest(dt2,dt1).astype(int)
        ind3 = find_nearest(dt3,dt1).astype(int)
        ind = (abs(dt2[ind2]-dt1) < upper_limit) * (abs(dt3[ind3]-
dt1)<upper_limit)*np.arange(1,len(dt1)+1)
        ind = ind[ind>0]-1
        ind = np.unique(ind).astype(int)
        ind2 = find_nearest(dt2,dt1[ind]).astype(int)
        ind3 = find_nearest(dt3,dt1[ind]).astype(int)

        dd2 = dt2[ind2] - dt1[ind]
        diff2 = 0
        if (len(dd2)>0):
            diff2 = sum(dd2)/len(dd2)
        t2 = t2 - diff2


        dd3 = dt3[ind3] - dt1[ind]
        diff3 = 0
        if (len(dd3)>0):
            diff3 = sum(dd3)/len(dd3)
        t3 = t3 - diff3
```

```
    new_time1.append(t1)
    new_time2.append(t2)
    new_time3.append(t3)

    new_data1.append(data1)
    new_data2.append(data2)
    new_data3.append(data3)

    new_time_acc.append(t_a)
    new_data_x_acc.append(data_x_acc)
    new_data_y_acc.append(data_y_acc)
    new_data_z_acc.append(data_z_acc)

    if (show_plot ):
        dt1 = t1[bio1]
        dt2 = t2[bio2]
        dt3 = t3[bio3]
        dh1 = w1[bio1]
        dh2 = w2[bio2]
        dh3 = w3[bio3]

        ind1 =
((t1>=a)*(t1<=b)*np.arange(1,len(t1)+1)).astype(int)
        ind2 =
((t2>=a)*(t2<=b)*np.arange(1,len(t2)+1)).astype(int)
        ind3 =
((t3>=a)*(t3<=b)*np.arange(1,len(t3)+1)).astype(int)
        ind1 = ind1[ind1>0]-1
        ind2 = ind2[ind2>0]-1
        ind3 = ind3[ind3>0]-1

        ind_dt1 =
((dt1>=a)*(dt1<=b)*np.arange(1,len(dt1)+1)).astype(int)
        ind_dt2 =
((dt2>=a)*(dt2<=b)*np.arange(1,len(dt2)+1)).astype(int)
        ind_dt3 =
((dt3>=a)*(dt3<=b)*np.arange(1,len(dt3)+1)).astype(int)

        ind_dt1 = ind_dt1[ind_dt1>0]-1
        ind_dt2 = ind_dt2[ind_dt2>0]-1
        ind_dt3 = ind_dt3[ind_dt3>0]-1

        plt.plot(t1[ind1],w1[ind1],'r-')
        plt.plot(t2[ind2],w2[ind2],'b-')
        plt.plot(t3[ind3],w3[ind3],'g-')
        plt.show();

        ind1 = None
        ind2 = None
        ind3 = None
```

Liite 3. Datan siivous ja R-piikkien korostaminen

```
def create_peak(t,data, startx, starty, midlex, midley, endx,
endy):
    a = 0
    b = 3.14159265/2
    c = -3.14159265*3/2
    y0 = starty + np.sin( (t-startx)/(midlex-startx)*b  )*(midley-
starty)
    y1 = midley + (1-np.sin(c+ (t-midlex)/(endx-midlex)*b
))*(endy-midley)
    y = data - data*(t>=startx)*(t<=endx)
    y  = y + (t>=startx)*(t<=midlex)*y0
    y  = y +  (t>midlex)*(t<=endx)*y1
    return y

def clean_data(t1,ecg1,t2,ecg2,t3,ecg3):
    ew1 = normalization(ecg1)
    ew2 = normalization(ecg2)
    ew3 = normalization(ecg3)

    (mt, mecg) =  weighted_median(t1, ew1, t2, ew2, t3, ew3)
    signal4 = nk2.ecg_clean(mecg,
sampling_rate=256,method='biosppy')
    mecg = scale_data4(medianf(signal4))
    sign1,info =
nk2.ecg_process(ecg1,sampling_rate=256,method='neurokit')
    sign2,info =
nk2.ecg_process(ecg2,sampling_rate=256,method='neurokit')
    sign3,info =
nk2.ecg_process(ecg3,sampling_rate=256,method='neurokit')

    i1 = range(0,len(sign1))
    rind1 = np.array(i1*sign1['ECG_R_Peaks']).astype(int)
    rind1 = rind1[rind1>0]
    j = np.arange(0,len(ecg1)).astype(int)
    step = 3
    for i in range(0, len(rind1)):
        x = rind1[i]
        M =max( max(ew1[x-10:x+10]), max(ew2[x-10: x+10]),
max(ew3[x-10: x+10]))
        ax = x-step
        ay = mecg[ax]
        bx = x + step
        by = mecg[bx]
        mecg = create_peak(j,mecg, ax, ay, x, M, bx, by)
        mecg[x]=M
    return (mt,mecg)
```

Liite 4. Painotettu mediaani

```
def sigma(d1, d2, d3):
    p0 = (d1>=0)*(d2>=0)*(d3>=0) * 1
    p1 = (d1>=0)*(d2>=0)*(1-p0)*1
    p2 = (d1>=0)*(d3>=0)*(1-p0)*1
    p3 = (d2>=0)*(d3>=0)*(1-p0)*1

    n0 = (d1<0)*(d2<0)*(d3<0)*1
    n1 = (d1<0)*(d2<0)*(1-n0)*1
    n2 = (d1<0)*(d3<0)*(1-n0)*1
    n3 = (d2<0)*(d3<0)*(1-n0)*1

    return (p0+p1+p2+p3-n0-n1-n2-n3)

def find_nearest(array, value):
    array = np.asarray(array)

    idx = np.array([])
    for v in value:
        i = (np.abs(array - v)).argmin()
        idx = np.append(idx,i)
    return idx

def weighted_median(t1, ecg1, t2, ecg2, t3, ecg3):
    di1 = range(0,len(t1))
    di2 = find_nearest(t2,t1).astype(int)
    di3 = find_nearest(t3,t1).astype(int)

    e1 = ecg1[di1]
    e2 = ecg2[di2]
    e3 = ecg3[di3]

    M = np.max((abs(e1), abs(e2), abs(e3)),axis=0)
    m = np.min((abs(e1),abs(e2),abs(e3)), axis=0)
    G = np.median((abs(e1), abs(e2), abs(e3)), axis=0)

    time = t1[di1]
    V = sigma(e1,e2,e3) * (4*G+5*M+m)/10.0
    return (time,V)
```

Liite 6. ECG datan jaottelu koulutus, validointi ja testidataan

```
def dataAreaIsFree(sample, device, start, sample_array,
device_array, index_array, size):
    for i in range(0, len(sample_array)):
        if (sample == sample_array[i] and device==device_array[i]
and start>=index_array[i]-int(size*0.8) and
start<=index_array[i]+size):
            return False
    return True

def fix2(data):
    data = data - np.median(data)
    data = data / max(abs(data))*100
    return data

training_sample = 20000
end_point_index = 256
validation_size = 40
test_size = 40

training_data1 = np.array([])
labeled_data1 = np.array([])
validation_data1 = np.array([])
validation_expected1 = np.array([])
test_data1 = np.array([])
test_label_data1 = np.array([])

N = validation_size + test_size
PERCENT = 0
sample_array = np.array([])
device_array = np.array([])
device_select = np.array([])
index_array = np.array([])

i=0
while (i<N):
    sample = random.randint(1,13)
    sample_data =
training_data_df[training_data_df['sample']==sample]
    SN = len(sample_data)
    device = random.randint(0,4) + 4
    index  = random.randint(0,SN-end_point_index-1)
    ecg2 = sample_data[field_names[device]].to_numpy()
    ecg2 = ecg2[index:index+end_point_index]
    if ( dataAreaIsFree(sample, device, index, sample_array,
device_array, index_array, end_point_index) and not
np.isnan(ecg2).any() ):

        device_select = np.append(device_select, device)
        for k in range(0,4):
            sample_array = np.append(sample_array,sample)
            device_array = np.append(device_array,(4 + k))
            index_array = np.append(index_array, index)
        i = i +1
sample_array = sample_array.astype(int)
device_array = device_array.astype(int)
```

```
index_array = index_array.astype(int)

M = N + training_sample
i = 0
while (i<M):
    percent = 100.0 * i / (1.0*M)
    if (percent > PERCENT):
        print(str(PERCENT)+ " %")
        PERCENT = PERCENT + 10
    sample = -1
    device = -1
    index = -1
    sample_data = None
    if (i<N):
        k=4*i
        sample = sample_array[k].astype(int)
        device = device_select[i].astype(int)
        index  = index_array[k].astype(int)
        sample_data =
training_data_df[training_data_df['sample']==sample]
    else:
        cont = True
        while (cont):
            sample = random.randint(1,13)
            sample_data =
training_data_df[training_data_df['sample']==sample]
            SN = len(sample_data)
            device = random.randint(0,4) + 4
            index  = random.randint(0,SN-end_point_index-1)
            if ( dataAreaIsFree(sample, device, index,
sample_array, device_array, index_array, end_point_index) and not
sample_data.isnull().values.any() ):
                cont=False

    ecg2 = sample_data[field_names[device]].to_numpy()
    acc_x = sample_data[field_names[ 9]].to_numpy()
    acc_y = sample_data[field_names[10]].to_numpy()
    acc_z = sample_data[field_names[11]].to_numpy()
    filtered_data = sample_data['filtered'].to_numpy()

    ecg2 = scale_data4(ecg2[index:index + end_point_index])
    ecg2 = fix2(ecg2)
    acc_x = acc_x[index:index + end_point_index]
    acc_y = acc_y[index:index + end_point_index]
    acc_z = acc_z[index:index + end_point_index]

    filtered_data = filtered_data[index:index + end_point_index]
    filtered_data = fix2(filtered_data)

    data = np.concatenate([[ecg2], [acc_x], [acc_y],[acc_z]])

    cont = not (np.isnan(filtered_data).any() or
np.isnan(data).any())
    if (cont == False):
        ic("Nan removed")

    if (cont==True and i<N):
        if (i<validation_size):
            validation_data1 = concatData(validation_data1, data)
```

```
            validation_expected1  =
concatData(validation_expected1, filtered_data)
            i = i + 1
        else:
            test_data1 = concatData(test_data1, data)
            test_label_data1 = concatData(test_label_data1,
filtered_data)
            i = i + 1
    else:
        training_data1 = concatData(training_data1, data)
        labeled_data1  = concatData(labeled_data1, filtered_data)
        i = i +1
```

Liite 7. Neuroverkon malli

```
model = Sequential()
model.add(Dense(256,input_shape=(4,256)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Dense(256, activation=LeakyReLU(alpha=0.01)))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(256))

model.compile(
    loss= 'mse',
    optimizer='sgd',
    metrics=[
        'accuracy',
        'MeanSquaredError'
    ]
)
```